

IMAGE ALGEBRA TECHNIQUES FOR PARALLEL COMPUTATION
OF THE
DISCRETE FOURIER TRANSFORM AND GENERAL LINEAR TRANSFORMS

By

PAUL D. GADER

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1986

Copyright 1986

by

Paul D. Gader

ACKNOWLEDGEMENTS

I would like to thank my wife, Ms. Elizabeth Dunn, for trying to understand me throughout this period, for not complaining about the disruption this dissertation has caused her, and for all the help that she has given me. I am grateful to my advisor, Dr. Gerhard X. Ritter, for teaching me so much about doing research in so little time, for allowing me freedom to do independent research on his contract, and for giving me the opportunity of working as a research assistant. To Drs. Joseph Wilson and Maurice Shrader-Frechette, I extend my deepest gratitude for helping me with computer science related matters so often. I am also indebted to all the members of my committee for the help and encouragement that they have given me over the years. I thank my father, Mr. Carl Gader, for all his support during my years in college. I acknowledge Drs. Dwayne Small and George Coyne for getting me started studying mathematics. I would also like to thank Dr. Sam Lambert and Mr. Neal Urquhart of the Air Force Armament Laboratory and Dr. Jasper Lupo of DARPA for partial support of this research under Contract F08635-84-C-0295.

Finally, I acknowledge my debt to the American taxpayers without whose support, through the G. I. Bill, federal student loans, grants, and work-study programs, and state and federally sponsored teaching and research assistantships, I would surely have never written this dissertation. I hope that I will contribute to society in such a way that everyone's support will be justified.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
LIST OF SYMBOLS	vi
ABSTRACT	viii
INTRODUCTION	1
Background of Image Algebra	2
Parallel Image Processing	4
Summary of Results Obtained	6
PART I. TEMPLATE DECOMPOSITIONS AND THE LINEAR SUBALGEBRA OF AN IMAGE ALGEBRA	11
CHAPTERS	
1. IMAGE ALGEBRA	12
1.1. Basic Definitions and Notation	12
1.2. Image Algebra and Linear Algebra	18
2. TEMPLATE DECOMPOSITIONS	25
2.1. Definitions and Background	26
2.2. Necessary and Sufficient Conditions for the Existence of Local Decompositions	30
3. TRANSLATION INVARIANT AND CIRCULANT TEMPLATES AND GENERALIZATIONS	46
3.1. Basic Definitions and Relationships	47
3.2. Circulant Templates and Polynomials	57
3.3. G-Templates	67
PART II. LOCAL DECOMPOSITIONS OF FOURIER TEMPLATES	83
4. TRIDIAGONAL DECOMPOSITIONS OF FOURIER MATRICES BASED ON MATRIX ALGEBRA ASSOCIATED WITH THE FFT	85
4.1. Derivation of FFT-Based Decompositions	86
4.2. Complexity of the FFT-Based Method	100

5. TRIDIAGONAL DECOMPOSITIONS OF FOURIER MATRICES BY OBLIQUE ELIMINATION	119
5.1. Background and Description of Oblique Elimination	120
5.2. Necessary and Sufficient Conditions for the Minimal Variable Solution to Exist	125
5.3. Application of Minimal Variable Oblique Elimination to the Fourier Matrices	133
5.4. Complexity Considerations	139
CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH	146
APPENDIX.	151
REFERENCES	109
BIOGRAPHICAL SKETCH	214

LIST OF SYMBOLS

\mathbf{Z} : The integers.

\mathbf{Z}_n : The integers modulo n .

\mathbf{Z}^k : The k -fold Cartesian product of the integers.

\mathbf{R} : The real numbers.

\mathbf{C} : The complex numbers.

F : A field (either \mathbf{R} or \mathbf{C}).

G : A finite group.

$\mathbf{C}[G]$: The group algebra of G over \mathbf{C} .

\mathbf{X} : A finite subset of \mathbf{Z}^k or, in particular, an $m \times n$ array of coordinates.

$\mathbf{x}, \mathbf{y}, \mathbf{z}$: Elements of \mathbf{X} .

$F^{\mathbf{X}}$: The algebra of images on \mathbf{X} with values in F .

$T_{\mathbf{X}}$: The set of templates on \mathbf{X} .

$L_{\mathbf{X}}$: The algebra of templates on \mathbf{X} with minimal configuration.

$C_{\mathbf{X}}$: The algebra of circulant templates on \mathbf{X} .

$G_{\mathbf{X}}$: The algebra of G -templates on \mathbf{X} .

O : The identity template of $L_{\mathbf{X}}$ under addition.

E : The identity template of $L_{\mathbf{X}}$ under \oplus .

R : A template configuration function.

T : A template.

$M_n(F)$: The algebra of $n \times n$ matrices over F .

Ψ : Isomorphism from $M_n(F)$ onto L_X .

$V(D)$: The vertex set of the digraph D .

$E(D)$: The arc set of the digraph D .

$D(R)$: The digraph associated with the template configuration R .

$D_\Delta(G)$: The Cayley color graph of G constructed from the set of generators Δ .

$\text{circ}(c_0, c_1, \dots, c_{n-1})$: An $n \times n$ circulant matrix.

DFT : The discrete Fourier transform.

FFT : Fast Fourier transform.

F_n : One-dimensional Fourier matrix of order n .

F_{mn} : Two-dimensional Fourier matrix of order $m \times n$.

$\omega_n = \exp[-2\pi i/n]$ where $i = \sqrt{-1}$.

$P(m,n)$: Permutation matrix corresponding to the permutation $a+bm \rightarrow an+b$ of \mathbf{Z}_{mn} .

$D(m,n)$: Twiddle factor matrices.

Σ_n : The symmetric group on n objects.

$\Gamma_z(T)$: The polynomial associated with the circulant template T at the point z .

$p_t(x,y,z)$: Same as $\Gamma_z(T)$.

Γ : Isomorphism from G_X onto $\mathbf{C}[G]$.

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

IMAGE ALGEBRA TECHNIQUES FOR PARALLEL COMPUTATION
OF THE
DISCRETE FOURIER TRANSFORM AND GENERAL LINEAR TRANSFORMS

By

Paul D. Gader

August 1986

Chairman: Dr. Gerhard X. Ritter
Major Department: Mathematics

This dissertation is the result of work done in conjunction with a research project concerning the development of an image algebra, an algebraic structure for digital image processing. A description of the use of the image algebra as a model and tool for the development of local, parallel algorithms is presented. Specifically, the image algebra is used as a model for local computation of linear transformations, and algorithms for local computation of the discrete Fourier transform are developed using traditional fast transform methods as well as numerical techniques.

The basic operators and operands of an image algebra are defined. Relationships between the image algebra and matrix algebra are described. Networks of processors are modeled as directed graphs. Images are represented as functions defined on the nodes of

a directed graph. It is shown that every linear image to image transform can be factored into a product of linear transformations which are compatible with the network if and only if the directed graph is strongly connected. An algebraic model of linear computations on Cayley networks is constructed. It is shown that the set of linear transformations which are translation invariant with respect to a Cayley network constructed from a finite group, G , is isomorphic to the group algebra of G over the complex numbers. Parallel algorithms for implementing discrete Fourier transforms of arbitrary size on mesh-connected arrays are derived using two different methods. One method is based on matrix algebra associated with fast Fourier transforms. The other method is based on a numerical technique for factoring square matrices into products of tridiagonal matrices.

INTRODUCTION

This dissertation is the result of work done in conjunction with a research project concerning the development of an image algebra, an algebraic structure for use in digital image processing. A description of the use of image algebra as a model and tool for the development of local, parallel algorithms is presented. Specifically, we use image algebra as a model for local computation of all linear transforms and we develop algorithms for the local computation of the discrete Fourier transform (DFT).

An image algebra is a heterogeneous algebraic structure whose operands are those objects commonly encountered in digital image processing and whose operators reflect the types of transformations commonly used in digital image processing [5]. There are two major reasons for developing an image algebra; an image algebra can provide a standard mathematical structure for expressing and investigating problems in digital image processing and can serve as the basis for an algebraically based, high level programming language. It is natural to use an algebraic structure as the basis for a programming language. There is beginning to be an awareness, however, not only in the image processing community, that traditional algebraic structures do not accurately reflect the representation and manipulation of operands as performed by digital computers. Thus, research has been initiated that focuses on developing new algebraic systems that accurately model the "environment" of the digital computer [7,13,33,45,50].

Upon reflecting on the history of mathematics, one sees that many of the advances and increases in understanding of mathematics and its applications were not possible

without the development of good notational systems. Thus, the possibility that this type of research will result in increased understanding of the mathematical problems in digital image processing is supported by historical precedent.

Background of Image Algebra

The use of image algebra in digital image processing was initiated, apparently independently, by Serra [52], Miller [32, 33], and Sternberg [55, 56, 58]. Ritter showed that the algebras used by these researchers were all equivalent and based on the operations of Minkowski addition and subtraction of sets in \mathbf{R}^n [44]. These operations are defined by

$$A + B = \{ a + b : a \in A, b \in B \}$$

and

$$A - B = \overline{\overline{A} + B}$$

where A and B are subsets of \mathbf{R}^n and the bar represents set complementation [2]. The Minkowski operations are sometimes referred to as the opening and closing or erosion and dilation operations.

A surprising number of techniques for extracting information from digital images can be developed using these operations. Many of these techniques can be found in the references already cited. Indeed, Sternberg designed and built a special purpose computer, called the Cytocomputer, to implement the Minkowski operations [30, 57].

The Minkowski operations are limited in scope to such a degree, however, that they cannot be used as the basis of a general purpose algebraically based language for digital image processing. Many common image processing techniques cannot be expressed using them [33]. Aware of a desire within the image processing community for an algebraic

structure that was capable of expressing most, if not all, of the common image processing algorithms, Ritter began the development of a more general image algebra [7, 44, 45, 46]. The research in this dissertation grew out of this effort to develop such an algebraic structure.

To insure that the image algebra developed be adequate for the intended uses, the following goals (among others) were set:

1.) Define a complete algebra, that is, one that is capable of expressing most, if not all, of the techniques being used in digital image processing.

2.) Define a simple algebra. The notation should be easy to read and understand by programmers who may not have an extensive mathematical background. There should be a small number of operators and they should reflect the structure of the computations.

3.) Develop basic algebraic properties and relations within the algebra.

4.) Determine relationships between the image algebra and existing algebraic structures.

5.) Demonstrate the applicability of the image algebra to the area of parallel image processing.

In this dissertation, we mainly address goals 4.) and 5.). We remark that the development of an image algebra is an ongoing process at this time.

We now present some background information about the area of parallel computer architectures in image processing.

Parallel Image Processing

Computer processing of digital images requires enormous amounts of computation. Moreover, in many cases the computations are “local” and translation invariant. Thus, the use of parallel computers in this area is highly desirable [13].

A digital image can be thought of as a function defined on the discrete rectangle

$$\mathbf{X} = \{ (i,j) : 0 \leq i \leq m-1, 0 \leq j \leq n-1 \}.$$

An image to image transformation is a mapping with domain and range contained in the set of all such functions. For many useful image to image transformations, it happens that the new value at a point $\mathbf{x} \in \mathbf{X}$ depends only on the values in a *neighborhood* of the point. Typical neighborhoods are the von Neumann and Moore neighborhoods depicted in Figures 1 and 2.

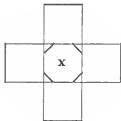


Figure 1. The von Neumann neighborhood.

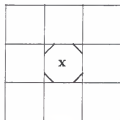


Figure 2. The Moore neighborhood.

Due to advances in VLSI (Very Large Scale Integrated Circuit) technology, it has become feasible to build large rectangular arrays of simple processors. The processors in these arrays each have their own small memory and can access the memory of the processors in their neighborhoods, typically von Neumann or Moore neighborhoods. Thus, these processors can compute functions of the values in their neighborhoods. An image to image transformation of this type is called a *local* transformation. Some of the arrays that have been built according to this design are the Massively Parallel Processor (MPP) [13, 41, 60], the Distributed Array Processor (ICL DAP) [20, 36], the Geometric Arithmetic Parallel Processor (GAPP) [8, 54], and the CLIP4 [12, 13]. These arrays of processors are generally referred to as massively parallel processors, cellular array processors, or mesh-connected arrays. The term cellular array is due to the fact that the design of mesh-connected arrays is based on the concept of a cellular automaton [35, 65].

There are other types of parallel computer architectures in use in digital image processing. The Cytocomputer, mentioned previously, is an example of a *pipeline* computer. The image is fed through a pipeline of processors, each processor being connected to a certain number of the others (its neighbors) and computing some function of the values in its neighborhood. A computer of this type implements translation invariant transformations; that is, the same transformation is applied to each point of the image.

A *systolic array* is a system of processors designed to perform a specific function or class of functions, such as matrix multiplication or convolutions. The simplest description is that data are fed into the array in a specified format, “flow” through the system, and the answer comes out. However, there are varying degrees of flexibility in systolic arrays [14].

Many other architectures have been proposed for image processing, some different from those described above and some variations of the previously described architectures [13, 49, 64]. A key feature of most of these architectures is that they have a large number of processors that communicate directly with only a small subset of the others. Many of them compute only translation invariant transformations and some of them have directional flows associated with the data.

There is a desire within the image processing community to use computers with parallel architectures as general purpose image processing computers. The heavy use of local transformations and the large computational burden make such architectures attractive. However, a good deal of research needs to be done to develop methods for using these architectures efficiently [43]. In particular, it has been pointed out by Schwartz that the lack of efficient methods for computing global linear transforms is an obstacle to the goal of developing real-time algorithms for scene analysis by robots [51].

Summary and Background of Results Obtained.

We outline the research described in this dissertation and the results that we have obtained. We also present some background information related to the specific results.

The major results of this dissertation are as follows:

- 1.) The establishment of necessary and sufficient conditions on the configuration structure of a network for the existence of local decompositions of all linear transforms.
- 2.) The definition and generalization of circulant templates. Establishment of relationships between circulant templates and other algebraic objects and application of these relationships to parallel computation of convolutions. We show that the set of all templates that are translation invariant with respect to a Cayley network constructed

using a group G is isomorphic to the group algebra of G over the complex numbers. We establish an equivalence between the invertibility of these templates and the invertibility of the discrete Radon transform on finite groups.

3.) The development of local decompositions of discrete Fourier transforms using matrix algebra associated with fast Fourier transforms (FFTs).

4.) Derivation of a numerical technique for factoring square matrices into products of tridiagonal matrices. Establishment of necessary and sufficient conditions for the technique to be successful. Development of local decompositions of discrete Fourier transforms using the technique.

5.) Implementation of parallel algorithms for computing DFTs locally using an image algebra preprocessor.

In Chapter 1, we define an image algebra and describe relationships between image algebra and linear algebra. In Chapter 2, we develop necessary and sufficient conditions on the configuration structure of networks of processors to ensure that every linear transform has a local decomposition with respect to the array. In Chapter 3, we define translation invariant and circulant templates and develop basic properties and relationships related to them. We generalize these notions to include the notion of templates that are translation invariant with respect to Cayley networks. In Chapter 4, we use matrix algebra associated with FFTs to develop local decompositions of discrete Fourier transforms of arbitrary size. These decompositions yield algorithms for implementing FFTs locally with respect to mesh-connected arrays. We provide a table which shows the estimated number of arithmetic and data shuffling steps required to implement these algorithms in certain cases. In Chapter 5, we use numerical linear algebra to derive an algorithm for computing tridiagonal decompositions of Fourier transforms. These

decompositions also yield algorithms for implementing FFTs locally with respect to mesh-connected arrays. In the Appendix, we present computer programs which are related to these decompositions. In particular, two of the programs are implementations of the 100×100 DFT using the local algorithms developed in Chapters 4 and 5. These programs, if implemented on a mesh-connected array having one processor per pixel, would take, at most, 18 parallel multiplication steps, 52 parallel addition steps, and 690 parallel permutation steps or 36 parallel multiplication steps, 36 parallel addition steps, and 654 parallel permutation steps. By a parallel permutation step, we mean switching the data in horizontally or vertically adjacent processors. These two programs are written in an extended version of FORTRAN 77 which accepts image algebra operands and operations and are therefore written in a parallel fashion although they were run on a serial machine.

The results in Chapter 2 are generalizations of theorems proven by Tchuenté [61]. The results in Chapter 3 were influenced by various sources. Translation invariant templates occur naturally in digital image processing. It is common practice to use circular convolutions to implement translation invariant transformations since circular convolutions can be computed using fast transform methods. Circulant templates are used to implement these convolutions. The generalizations of circulants grew out of an awareness of the possible uses of Cayley networks as models for parallel computer architectures and the importance of translation invariant transformations in parallel processing; these generalizations were also influenced by observations that the discrete Fourier transform is related to the theory of group representations [3].

The material in Chapter 4 has an extensive background, since use is made of FFTs. Since we use matrix representations of FFTs, we relate this aspect of their history only.

Use of matrix algebra to describe FFTs seems to have originated with Good [16], who used them to demonstrate the computational effectiveness of an FFT algorithm seven years before the famous paper of Cooley and Tukey [9]. Pease derived tridiagonal decompositions of power-of-two Fourier matrices and pointed out their value in parallel processing [40]. Others have used matrix algebra to prove the correctness of variations of FFT algorithms [23, 63]. Good also used matrix algebra to compare different FFT algorithms [17]. Rose [48] assimilated the various matrix representations and developed a number of matrix identities representing various techniques for implementing FFTs on serial processors. In a related paper, Parlett [39] shows that the Winograd FFTs [67] can be represented as eigenvalue-eigenvector decompositions of circulant matrices. Radix-two FFTs have been implemented on mesh-connected arrays by Jesshope [21] and Strong [60]. These implementations are based on the decompositions developed by Pease. Due to image sensor characteristics, however, images may be digitized in a variety of dimensions, such as 120×360 [47]. Furthermore, architectural considerations sometimes result in mesh-connected arrays being built with dimensions other than powers of two. Examples of such arrays are those built using GAPP chips which have dimensions that are multiples of six [8, 54]. Moreover, even if it were possible on a mesh-connected array, experts in the field warn against enlarging the data set by appending enough zeroes to force power of two dimensions [18]. Hence there is a need for methods of implementing arbitrary discrete Fourier transforms locally, which we provide.

In Chapter 5, we develop alternative methods of computing local decompositions of Fourier matrices. The methods are numerical in nature and are (seemingly) unrelated to FFTs. They are based on some theoretical work done by Tchuente concerning factoring matrices into products of tridiagonal matrices [61]. Tchuente is concerned with the general problem of calculating linear transforms in parallel and makes no mention of any

specific transforms nor does he address implementation issues in much detail. We apply a special case of his general method to the Fourier matrices. We develop necessary and sufficient conditions for the method to work and use them to show that it can be applied to the Fourier matrices.

We conclude this introduction by pointing out that we expect this dissertation to be read and used by members of a multidisciplinary research team. We have written it with this in mind and therefore have attempted to include as much detail as possible to keep the results accessible to nonmathematicians. We hope that we have succeeded and that this work will be useful to others.

PART I

TEMPLATE DECOMPOSITIONS AND THE LINEAR SUBALGEBRA
OF AN IMAGE ALGEBRA

In Part I of this dissertation we consider some general questions concerning the linear subalgebra of an image algebra and the applications of this subalgebra as a mathematical model of parallel image processing.

Part I is divided into three chapters. In the first chapter, we define the operators and operands of image algebra and we describe the relationship between image algebra and linear algebra. In the second chapter, we show how the image algebra can be used as a mathematical model of parallel computer architectures. We pose the question of whether all linear transforms can be computed on a given architecture using the notion of local template decompositions. We develop necessary and sufficient conditions for the existence of local decompositions. In the third chapter, we examine two important classes of templates, translation invariant and circulant templates. These templates are used to implement convolutions in the image algebra and occur frequently in digital image processing. They are closely related to the discrete Fourier transform. In fact, any local decomposition of the Fourier transform yields a local decomposition of a circulant template. Local decompositions of the discrete Fourier transform are the subject of Part II of this dissertation. We generalize the notion of a circulant template by defining a larger class of templates which we call G-templates. We discuss possible applications of G-templates to parallel image processing.

CHAPTER 1

IMAGE ALGEBRA

In this chapter, we present the basic definitions and properties of an image algebra. We also describe the relationships between image algebra and linear algebra. The notation and relationships introduced here will be used throughout this dissertation.

1.1. Basic Definitions and Notation

An image algebra consists of a number of sets with various operations defined on and between the sets. The sets consist of a finite coordinate set, \mathbf{X} , and three sets of operands. The sets of operands are a field F (either the real or complex numbers), the set of graphs of all functions $a : \mathbf{X} \rightarrow F$ which we call images, denoted $F^{\mathbf{X}}$, and a set of objects called templates which are used to transform images. We consider F to be endowed with the usual field operations as well as the lattice operations of maximum and minimum. Operations on $F^{\mathbf{X}}$ are defined pointwise using operations on F . Operations are defined between templates and images. A template is defined relative to two coordinate sets, \mathbf{X} and \mathbf{Y} , and induces three different mappings from $F^{\mathbf{X}}$ to $F^{\mathbf{Y}}$, one for each operation between templates and images. Operations are also defined between templates, some pointwise and some as generalizations of convolutions. The definitions of these operations are designed to reflect the types of computations and computing environments which are currently encountered, or likely to be encountered in the near future, in digital image processing. We now give the formal definitions.

1.1.1. Operands

Throughout this dissertation \mathbf{C} , \mathbf{R} , and \mathbf{Z} shall denote the sets of complex numbers, real numbers, and integers respectively, and, unless specifically stated otherwise, \mathbf{X} denotes a finite subset of \mathbf{Z}^k where \mathbf{Z}^k denotes the k -fold Cartesian product of \mathbf{Z} .

We make a linearly ordered field of \mathbf{C} by defining $a + bi \leq c + di$ if and only if $a < c$ or $a = c$ and $b \leq d$. With this ordering the maximum and minimum of two complex numbers can be defined. We denote the binary operations of maximum and minimum on the set of real or complex numbers by \vee and \wedge , respectively. Henceforth, we shall use F to denote either the field of complex numbers or the real numbers, the complex numbers being ordered as above and the real numbers being ordered in the usual way.

Definition 1.1. An *image*, A , on \mathbf{X} with values in F is the graph of a function $a : \mathbf{X} \rightarrow F$; that is, $A = \{ (x, a(x)) : x \in \mathbf{X} \}$. The set of all images on \mathbf{X} with values in F is denoted by $F^{\mathbf{X}}$. The set \mathbf{X} is called the set of *image coordinates*, or *coordinate set*. When F is understood we say that A is an image on \mathbf{X} or that A is an image.

Definition 1.2. Let \mathbf{X} , \mathbf{Y} be finite subsets of \mathbf{Z}^k and \mathbf{Z}^n , respectively. A *template* from \mathbf{Y} to \mathbf{X} is a pair (T, T) where

- 1.) $T : \mathbf{Y} \rightarrow 2^{\mathbf{X}}$, with $2^{\mathbf{X}}$ denoting the power set of \mathbf{X} , and
- 2.) $T : \mathbf{Y} \rightarrow F^{\mathbf{X}}$ such that

$$T(y) \equiv \{ (x, t_y(x)) : t_y(x) = 0 \text{ if } x \notin T(y), x \in \mathbf{X} \}.$$

That is, $T(y)$ is the graph of a function $t_y : \mathbf{X} \rightarrow F$ whose support lies in $T(y)$. The point y is called the *center* of the set $T(y)$, and the values $t_y(x)$ for $x \in T(y)$ are called

the *weights* of $T(y)$.

If (T, T) is a template, then T is called a *template function with configuration T* , and T is called a *template configuration, neighborhood configuration*, or just *configuration*, for \mathbf{Y} on \mathbf{X} . If $\mathbf{Y} = \mathbf{X}$ then (T, T) is called a *template on \mathbf{X}* and T a *template or neighborhood configuration on \mathbf{X}* . Whenever it is not necessary to specify T explicitly, we simply say that T is a template (to distinguish it from “ T is a function from \mathbf{Y} to $F^{\mathbf{X}}$ ”). The set of all templates from \mathbf{Y} to \mathbf{X} will be denoted by $T_{\mathbf{Y},\mathbf{X}}$ and if $\mathbf{Y} = \mathbf{X}$, then we define $T_{\mathbf{X}} \equiv T_{\mathbf{Y},\mathbf{X}}$.

We usually use the conventions that $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbf{X}$, A, B, C are images which are the graphs of the functions $a, b, c : \mathbf{X} \rightarrow F$ respectively, and R, S, T are templates with corresponding gray level functions $r_{\mathbf{x}}, s_{\mathbf{x}}$, and $t_{\mathbf{x}}$.

1.1.2. Image Operations

Let $A, B \in F^{\mathbf{X}}$. The binary operations of addition, multiplication, maximum, and exponentiation are defined as follows:

$$A + B \equiv \{ (x, c(x)) : c(x) = a(x) + b(x), x \in \mathbf{X} \}$$

$$A * B \equiv \{ (x, c(x)) : c(x) = a(x) * b(x), x \in \mathbf{X} \}$$

$$A \vee B \equiv \{ (x, c(x)) : c(x) = a(x) \vee b(x), x \in \mathbf{X} \}$$

$$A^B \equiv \{ (x, c(x)) : c(x) = a(x)^{b(x)} \text{ if } a(x) \neq 0, \text{ else } c(x) = 0, x \in \mathbf{X} \}.$$

If $F = \mathbf{R}$, we restrict the latter binary operation to those pairs of images A, B for which $a(x)^{b(x)} \in \mathbf{R}$ whenever $a(x) \neq 0$.

The *dot product* of two images is defined by

$$A \cdot B \equiv \sum_{x \in X} a(x)b(x) .$$

An image A is called a *constant image* if all its gray values are the same; that is, if $a(x) = k$ for some real number k and for all $x \in X$. The constant images O and I defined by $O \equiv \{(x,0): x \in X\}$ and $I \equiv \{(x,1): x \in X\}$ are the additive and multiplicative identities respectively of F^X .

Suppose $k \in F$ and A is a constant image with $a(x) = k$. Then we define $B^k \equiv B^A$, $kB \equiv A * B$, and $k+B \equiv A + B$. We note that exponentiation is defined even when $a(x) = 0$. Also, subtraction, division and minimum can be defined in terms of the basic operations and inverses. Specifically: $A - B \equiv A + (-B)$, $A/B \equiv A * B^{-1}$, and $A \wedge B \equiv -(A \vee -B)$.

Characteristic functions on images can be defined in terms of the previous binary operations. For example, if A and B are images on X , then the characteristic function of A greater than B is given by

$$c_{>B}(A) \equiv [(A-B) \vee 0]^{-1} * [(A-B) \vee 0] .$$

Thus,

$$c_{>B}(A) = \{ (x, c(x)) : c(x) = 1 \text{ if } a(x) > b(x), \text{ else } c(x) = 0 \} .$$

Whenever B is the constant image with gray values equal to k , it is customary to replace B by k in the above definition. The remaining characteristic functions of images can be defined in a similar fashion, using complementation and products.

The basic unary operations on F^X are the functions available in most high level programming languages. In fact, any function $f: F \rightarrow F$ induces a function $f: F^X \rightarrow F^X$ by $f(A) = \{ (x, c(x)) : c(x) = f(a(x)) \} ^\wedge$.

1.1.3. Image-Template Operations

There are three operations between images and templates which are used to transform an image. They are denoted \oplus , \odot , and \boxplus . These neighborhood operations transform each image point by performing the basic operation of addition or maximum on a weighted collection of neighboring image values. In particular, if $A \in F^X$ and $T \in T_{Y,X}$, then

$$A \oplus T \equiv \{ (y, c(y)) : c(y) = \sum_{x \in T(y)} a(x) \cdot t_y(x), y \in Y \}$$

$$A \odot T \equiv \{ (y, c(y)) : c(y) = \bigvee_{x \in T(y)} a(x) \cdot t_y(x), y \in Y \}$$

$$A \boxplus T \equiv \{ (y, c(y)) : c(y) = \bigvee_{x \in T(y)} a(x) + t_y(x), y \in Y \}.$$

Note that $A \in F^X$, while $A \oplus T \in F^Y$. Thus, template operations on images provide a tool for image rotation, zooming, image reduction, masked extraction, and so on. In this dissertation, however, we restrict ourselves to templates on X .

1.1.4. Template Operations

We first define the pointwise operations. The basic operations of addition, multiplication and maximum on the set of templates are induced pointwise by the corresponding operations of F^X . In particular, if (T, T) and (S, S) are elements of T_X , then the sum, product, and maximum of (T, T) and (S, S) are given by

$$(T, T) + (S, S) = (R, R)$$

where

$$R(x) \equiv T(x) + S(x) \quad \text{and} \quad R(x) \equiv T(x) \cup S(x),$$

$$(T, T) * (S, S) = (R, R)$$

where

$$R(\mathbf{x}) \equiv T(\mathbf{x}) * S(\mathbf{x}) \quad \text{and} \quad R(\mathbf{x}) \equiv T(\mathbf{x}) \cup S(\mathbf{x}),$$

and

$$(T, T) \vee (S, S) = (R, R)$$

where

$$R(\mathbf{x}) \equiv T(\mathbf{x}) \vee S(\mathbf{x}) \quad \text{and} \quad R(\mathbf{x}) \equiv T(\mathbf{x}) \cup S(\mathbf{x}).$$

Again, subtraction, division, minimum, and scalar multiplication can be derived from these basic operations in a straightforward manner. Specifically, $T - S$ is defined by $(T - S)(\mathbf{x}) = T(\mathbf{x}) - S(\mathbf{x})$, T/S by $(T/S)(\mathbf{x}) = T(\mathbf{x})/S(\mathbf{x})$, $T \wedge S$ by $(T \wedge S)(\mathbf{x}) = T(\mathbf{x}) \wedge S(\mathbf{x})$ and rT by $(rT)(\mathbf{x}) = rT(\mathbf{x})$. The configurations resulting from these operations are the same as the configurations resulting from the operations from which they were derived.

We now define the generalized convolutions between templates. Let $(S, S), (T, T) \in T_X$. Then

$(S, S) \oplus (T, T) = (R, R)$ is defined by

$$R(\mathbf{x}) \equiv (S \oplus T)(\mathbf{x}) \equiv \{ (\mathbf{z}, r_{\mathbf{x}}(\mathbf{z})) : r_{\mathbf{x}}(\mathbf{z}) = \sum_{\mathbf{y} \in T(\mathbf{x}) : \mathbf{z} \in S(\mathbf{y})} t_{\mathbf{x}}(\mathbf{y}) s_{\mathbf{y}}(\mathbf{z}), \mathbf{z} \in \mathbf{X} \},$$

$$R(\mathbf{x}) \equiv \bigcup_{\mathbf{y} \in T(\mathbf{x})} S(\mathbf{y}), \quad \text{and} \quad r_{\mathbf{x}}(\mathbf{z}) = 0 \text{ if } \mathbf{z} \notin R(\mathbf{x});$$

$(S, S) \boxtimes (T, T) = (R, R)$ is defined by

$$R(\mathbf{x}) \equiv (S \boxtimes T)(\mathbf{x}) \equiv \{ (\mathbf{z}, r_{\mathbf{x}}(\mathbf{z})) : r_{\mathbf{x}}(\mathbf{z}) = \bigvee_{\mathbf{y} \in T(\mathbf{x}) : \mathbf{z} \in S(\mathbf{y})} t_{\mathbf{x}}(\mathbf{y}) + s_{\mathbf{y}}(\mathbf{z}), \mathbf{z} \in \mathbf{X} \},$$

$$R(\mathbf{x}) \equiv \bigcup_{\mathbf{y} \in T(\mathbf{x})} S(\mathbf{y}), \quad \text{and} \quad r_{\mathbf{x}}(\mathbf{z}) = 0 \text{ if } \mathbf{z} \notin R(\mathbf{x});$$

$(S, S) \odot (T, T) = (R, R)$ is defined by

$$R(\mathbf{x}) \equiv (S \otimes T)(\mathbf{x}) \equiv \{ (z, r_x(z)) : r_x(z) = \bigvee_{y \in T(\mathbf{x}) : z \in S(y)} t_x(y) s_y(z), z \in X \},$$

$$R(\mathbf{x}) \equiv \bigcup_{y \in T(\mathbf{x})} S(y), \quad \text{and} \quad r_x(z) = 0 \text{ if } z \notin R(\mathbf{x}).$$

The notation $y \in T(\mathbf{x}) : z \in S(y)$ means that $y \in T(\mathbf{x})$ and $z \in S(y)$. Note that if $z \notin S(y)$, then $s_y(z) = 0$. Hence, in the case of \oplus we may write

$$r_x(z) = \sum_{y \in T(\mathbf{x})} t_x(y) s_y(z).$$

This concludes the brief description of the image algebra. It has been shown that this set of operands and operators is sufficient to express all “interesting” image to image transformations [46]. In the rest of this dissertation, we will focus on the linear subalgebra of the image algebra. In the next section we describe this subalgebra.

1.2. Image Algebra and Linear Algebra

In this section, we describe relationships between image algebra and linear algebra. These relationships will be used throughout this dissertation.

Definition 1.6. Define a relation, \sim , on T_X by declaring $S \sim T$ if and only if for every $\mathbf{x}, y \in X$, $t_x(y) = s_x(y)$.

The relation, \sim , is clearly an equivalence relation. Denote by L_X the set of equivalence classes of T_X under \sim and by $[T]$ a typical element of L_X . The proof of the next theorem is routine and therefore we omit it.

Theorem 1.7. If $S_1 \sim S_2$ and $T_1 \sim T_2$, then

$$1.) [T_1 + S_1] = [T_2 + S_2].$$

$$1.) [T_1 \oplus S_1] = [T_2 \oplus S_2].$$

$$3.) A \oplus T_1 = A \oplus T_2.$$

Hence, we can use the operations of T_X to define operations on L_X .

Definition 1.8. If $[T], [S] \in L_X$, $\bar{T} \in [T]$, and $\bar{S} \in [S]$, then define

$$1.) [T] + [S] \equiv [\bar{T} + \bar{S}]$$

$$2.) [T] \oplus [S] \equiv [\bar{T} \oplus \bar{S}]$$

$$3.) A \oplus [T] \equiv A \oplus \bar{T}.$$

If $[T] \in L_X$, then there exists a unique template $S \in [T]$ with the property that for every $R \in [T]$, $S(x) \subset R(x)$ for every $x \in X$. S has the property that $s_x(y) = 0$ if and only if $y \notin S(x)$. We say that such a template has *minimal configuration*.

Henceforth, we shall identify an equivalence class of templates with the representative of that equivalence class having minimal configuration. By Theorem 1.7, the algebraic operations remain valid. That is, if S and T are templates with minimal configuration, then $S + T$ and $S \oplus T$ are also template with minimal configuration when the operations are considered to be between equivalence classes. Note that the operations $+$ and \oplus on L_X are not the same as the operations $+$ and \oplus on T_X . For example, if $T \in T_X$ with nonempty minimal configuration, then $T - T$ does not have minimal configuration (in T_X). Computationally, however, T_X and L_X are essentially the same with respect to \oplus . From now on, we shall refer to the elements of L_X as templates. Note that when defining a template $T \in L_X$, there is no need to specify the configuration once the template function has been specified.

Denote by O and E the templates defined by

$$O(x) \equiv \{ (y, t_x(y)) : t_x(y) = 0 \text{ for every } x \in X \}$$

and

$$E(\mathbf{x}) \equiv \{ (y, e_x(y)) : e_x(\mathbf{x}) = 1 \text{ and } e_x(y) = 0 \text{ if } y \neq \mathbf{x} \}.$$

For every $T \in L_X$, $O + T = T + O = T$. Note that $S, T \in L_X$ and $S - T = O$ if and only if $S = T$. This is not true in T_X . The template E is an identity element for \oplus ; that is, $T \oplus E = E \oplus T = T$ and $A \oplus T = A$ for every $T \in L_X$ and $A \in F_X$.

Since X is a finite set, it can be linearly ordered. Thus, we can write $X = \{x_0, x_1, \dots, x_{n-1}\}$. We define a mapping $\nu : F^X \rightarrow F^n$ by $\nu(A) = (a(x_0), a(x_1), \dots, a(x_{n-1}))^t$. Since $\nu(rA + sB) = r\nu(A) + s\nu(B)$ and ν is 1-1 and onto, ν is a vector space isomorphism.

Let $(M_n, *, +)$ denote the ring of $n \times n$ matrices with entries from F under matrix multiplication and addition. For any $T \in L_X$, we define a matrix $M_T = (m_{ij})$ where $m_{ij} \equiv t_{x_i}(x_j)$. Note that the i^{th} row of M_T is $\nu(T(x_i))^t$. Define a mapping $\Psi : L_X \rightarrow M_n$ by $\Psi(T) = M_T$.

Theorem 1.9. Ψ is a ring isomorphism of $(L_X, \oplus, +)$ onto $(M_n, *, +)$. That is, if $S, T \in L_X$, then

- 1.) $\Psi(S+T) = \Psi(S) + \Psi(T)$ or $M_{S+T} = M_S + M_T$
- 2.) $\Psi(S \oplus T) = \Psi(T)\Psi(S)$ or $M_{S \oplus T} = M_T M_S$
- 3.) Ψ is 1-1 and onto
- 4.) $\Psi(E) = I_n$ where I_n denotes the $n \times n$ identity matrix
- 5.) $(L_X, \oplus, +)$ is a ring.

Proof.

- 1.) Let $S, T \in L_X$. Then

$$\Psi(S+T) = \begin{bmatrix} \nu((S+T)(\mathbf{x}_0))^t \\ \vdots \\ \nu((S+T)(\mathbf{x}_{n-1}))^t \end{bmatrix}$$

Since $\nu((S+T)(\mathbf{x}_i)) = \nu(S(\mathbf{x}_i)+T(\mathbf{x}_i)) = \nu(S(\mathbf{x}_i))+\nu(T(\mathbf{x}_i))$, we may conclude that $\Psi(S+T) = \Psi(S)+\Psi(T)$.

2.) Let $R = S \oplus T$. Then, by definition of \oplus , $r_{\mathbf{x}_i}(\mathbf{x}_j) = \sum_{\mathbf{y} \in T(\mathbf{x}_i)} t_{\mathbf{x}_i}(\mathbf{y})s_{\mathbf{y}}(\mathbf{x}_j)$. By

definition of matrix multiplication $M_T M_S = (c_{ij})$ where $c_{ij} = \sum_{k=0}^{n-1} t_{\mathbf{x}_i}(\mathbf{x}_k)s_{\mathbf{x}_k}(\mathbf{x}_j)$. Since $t_{\mathbf{x}_i}(\mathbf{x}_k) = 0$ if $\mathbf{x} \notin T(\mathbf{x}_i)$, we must have $c_{ij} = r_{\mathbf{x}_i}(\mathbf{x}_j)$ which implies that $\Psi(S \oplus T) = \Psi(T)\Psi(S)$.

3.) The map Ψ is clearly onto. Furthermore Ψ is 1-1 since if $\Psi(T) = \Psi(S)$ then $\Psi(T-S) = 0$. Recall that O is the only template with all zero gray values and minimal configuration. Hence $T-S = O$ so $T = S$.

4.) If $M_E = (m_{ij})$ then by definition $m_{ij} = 1$ if and only if $i = j$.

5.) The set L_X is an abelian group under addition since F^X is an abelian group under addition and addition on L_X is induced pointwise by F^X . Suppose that $R, S, T \in L_X$. Then $R \oplus (S \oplus T) = \Psi^{-1}(\Psi(R \oplus (S \oplus T))) = \Psi^{-1}(\Psi(S \oplus T) * \Psi(R)) = \Psi^{-1}(\Psi(T) * \Psi(S) * \Psi(R)) = (R \oplus S) \oplus T$. It is also true that $R \oplus (S+T) = \Psi^{-1}(\Psi(R \oplus (S+T))) = \Psi^{-1}((\Psi(S)+\Psi(T)) * \Psi(R)) = R \oplus S + R \oplus T$. Similarly $(S+T) \oplus R = S \oplus R + T \oplus R$.

Q.E.D.

Theorem 1.10. For every $A \in F^X$ and $T \in L_X$, $A \oplus T = \nu^{-1}(\Psi(T)\nu(A))$.

Proof.

Let $B = A \oplus T$. Then, by definition of \oplus , $b(x) = \sum_{y \in T(x)} a(y)t_x(y)$. Furthermore, $\Psi(T)\nu(A) = (c_0, c_1, \dots, c_{n-1})^t$ where $c_i = \sum_{k=0}^{n-1} t_{x_i}(x_k)a(x_k)$. Since $t_{x_i}(x_k) = 0$ if $x_k \notin T(x_i)$, we have that $c_i = b(x_i)$ which shows that $\nu(B) = \Psi(T)\nu(A)$. The conclusion follows from the fact that ν is invertible.

Q.E.D.

Note that there is a different isomorphism Ψ for every linear ordering on X . In the next theorem, we describe the relationship between them.

Denote by Σ_n the group of permutations on $\{0, 1, \dots, n-1\}$.

Definition 1.11. Let $\sigma \in \Sigma_n$ and define the $n \times n$ matrix P_σ by

$$P_\sigma = (p_{ij}) \text{ where } p_{ij} = \begin{cases} 1 & \text{if } j = \sigma(i) \\ 0 & \text{otherwise} \end{cases}.$$

P_σ is called a *permutation matrix*.

It is well known that permutation matrices are invertible and that $P_\sigma^{-1} = P_\sigma^t = P_{\sigma^{-1}}$. In fact, the set of all $n \times n$ permutation matrices forms a group which is isomorphic to Σ_n . Note that if $A = (a_{ij})$ is an $n \times n$ matrix, then multiplication on the left by P_σ permutes the rows of A by σ^{-1} and multiplication on the right by $P_{\sigma^{-1}} = P_\sigma^t$ permutes the columns of A by σ^{-1} . Hence, we can write $P_\sigma A P_\sigma^t = (a_{\sigma(i), \sigma(j)})$.

Theorem 1.12. The following two assertions hold for any X :

1.) Assume that $X_1 = \{x_0, x_1, \dots, x_{n-1}\}$ and $X_2 = \{y_0, y_1, \dots, y_{n-1}\}$ are two different orderings of X . Let $\Psi_1 : L_X \rightarrow M_n$ and $\Psi_2 : L_X \rightarrow M_n$ be defined relative to

X_1 and X_2 respectively. There exists a permutation matrix, P , such that for every $T \in L_X$, $\Psi_1(T) = P\Psi_2(T)P^t$.

2.) Conversely, assume that $\Psi_1 : L_X \rightarrow M_n$ and that there exists an $n \times n$ permutation matrix P with the property that for every $T \in L_X$ there exists a matrix $M(T) \in M_n$ such that $\Psi_1(T) = PM(T)P^t$. Then there exists an ordering on X such that if $\Psi_2 : L_X \rightarrow M_n$ is defined relative to that ordering, then $\Psi_2(T) = M(T)$ for every $T \in L_X$.

Proof.

1.) Let $\sigma \in \Sigma_n$ be the permutation defined by $x_{\sigma(i)} = y_i$ for $i \in [0, n-1]$. Denote $\Psi_1(T) = (\mu_{ij})$ and $\Psi_2(T) = (\gamma_{ij})$. Then $P_\sigma \Psi_1(T) P_\sigma^t = (\mu_{\sigma(i), \sigma(j)})$. Furthermore, $\gamma_{ij} = t_{y_i}(y_j) = t_{x_{\sigma(i)}}(x_{\sigma(j)}) = \mu_{\sigma(i), \sigma(j)}$, which implies that $\Psi_1(T) = P_\sigma^t \Psi_2(T) P_\sigma$.

2.) Assume that $X = \{x_0, x_1, \dots, x_{n-1}\}$ is the ordering on X used to define Ψ_1 . Since P is a permutation matrix, there exists a $\sigma \in \Sigma_n$ such that $P = P_\sigma$. Define a different ordering $\{y_0, y_1, \dots, y_{n-1}\}$ of X by taking $y_{\sigma(i)} \equiv x_i$ or $y_i = x_{\sigma^{-1}(i)}$. Let $\Psi_2 : L_X \rightarrow M_n$ be defined relative to this ordering. By 1.), $P^t \Psi_1(T) P = \Psi_2(T)$ for every $T \in L_X$. By assumption, $P^t \Psi_1(T) P = M(T)$ for every $T \in L_X$. Hence $\Psi_2(T) = M(T)$.

Q.E.D.

Henceforth, we shall use the symbol Ψ to denote a mapping $\Psi : L_X \rightarrow M_n$ as described in this section. The ordering on X will be understood as given and no mention of it will be made in general.

The special case that X is an $m \times n$ array is of particular interest. In this case, we take $X = \{(x, y) : 0 \leq x \leq m-1, 0 \leq y \leq n-1\}$. We can order X lexicographically in row major form by mapping $(x, y) \rightarrow xn + y$. The mappings ν and Ψ then have the following form:

$\nu(A) = (a_0, a_1, \dots, a_{nm-1})^t$ where $a_i = a(x, y)$ if $i = xn + y$ and

$$\Psi(T) = \begin{bmatrix} \nu(T(0,0))^t \\ \nu(T(0,1))^t \\ \vdots \\ \nu(T(m-1,n-1))^t \end{bmatrix}.$$

Henceforth, whenever \mathbf{X} is an $m \times n$ array, we shall assume that these particular mappings have been used.

It is clear from the results of this section that any relationship which can be expressed using the usual notation of finite-dimensional linear algebra can also be expressed using the notation of image algebra. The image algebra notation differs from conventional matrix-vector notation in that it reflects the way that computations are carried out in digital image processing. There is a great deal of knowledge expressed in terms of matrices and vectors. The relationships described in this section provide the link to this knowledge and offer direct methods for implementing linear image transforms on parallel computer architectures.

CHAPTER 2

TEMPLATE DECOMPOSITIONS

As mentioned in the Introduction, in order to take advantage of the capabilities of computers with parallel or distributed architectures, there is a need to understand the process of computing global linear transforms locally. In this chapter, we establish necessary and sufficient conditions for the existence of local decompositions of all linear transformations with respect to a directed network of processors. One particular implication of this theorem is that any linear transformation can be factored into a sequence of linear transformations, each of which is local with respect to a mesh-connected array. The theorem is much more general, however, implying that any linear transformation can be factored into a sequence of linear transformations, each of which is local with respect to the interconnection structure of a particular network of processors, as long as any processor in the network can communicate with any other processor in the network, that is, as long as there is a path through the network between every pair of processors. The theorem is an existence theorem and does not yield an efficient technique for computing such decompositions, although a constructive proof is given. The value of such a theorem is that it shows that such decompositions exist and, therefore, that it is not futile to attempt to develop methods for computing them.

Template configurations are used here to model processor arrays. We set up correspondences between template configurations, directed graphs, and networks of processors. A computation is considered to be compatible with a network if the templates required to implement the computation are local with respect to the configuration

modeling the network. We then consider the problem of factoring templates into products (with respect to \oplus) of templates which are local with respect to a given configuration. We show that every template has such a local decomposition if and only if the directed graph corresponding to the configuration is strongly connected. This is the major result of this chapter and is a generalization of a theorem proven by Tchuente which we will state using the notation provided by image algebra [61].

2.1. Definitions and Background

Throughout this chapter, let $X \subset Z^k$ be a finite set, $R : X \rightarrow 2^X$ an arbitrary template configuration with the property that $x \in R(x)$ for every $x \in X$, and assume that images have values in C . We think of the elements of X as being processors in a network. The assumption that $x \in R(x)$ can be interpreted as meaning that every processor in the network has direct access to its own memory. Similarly, the interpretation of $y \in R(x)$ is that processor x has direct access to the memory of processor y .

Definition 2.1. Let $T \in L_X$. A *template decomposition* of T is a set $\{T_i\}_{i=1}^n$ of templates $T_i \in L_X$ such that $T = \bigoplus_{i=1}^n T_i$. The T_i are called the *factors* of the decomposition. We write $T = \bigoplus_{i=1}^n T_i$ is a decomposition of T .

Definition 2.2. We say that a template $T \in L_X$ is *local* with respect to R if and only if $T(x) \subset R(x)$ for every $x \in X$. If R is understood, then we say that T is a local template.

Definition 2.3. Let $T \in L_X$ and assume that $\{T_i\}_{i=1}^n$ is a template decomposition of T . We say that $\{T_i\}_{i=1}^n$ is a *local decomposition* of T with respect to R if and only if each

T_i is local with respect to R for $i \in [1, n]$. If $T \in L_X$ and if there exists a decomposition $\{T_i\}_{i=1}^n$ of T which is local with respect to R , then we say that T has a local decomposition with respect to R . If R is understood, then we say that T has a local decomposition.

Definition 2.4. A *digraph*, or *directed graph*, is a pair $D = (V, E)$ where V is a finite set and $E \subset V \times V$. The elements of V are called *vertices*. If $(x, y) \in E$, then (x, y) is called an *arc* from x to y or just an arc. If there is a need, we may write $V = V(D)$ to emphasize that V is the vertex set for the digraph D ; we may also write $E = E(D)$.

Definition 2.5. A *graph* is a digraph $G = (V, E)$ with the additional property that if $(x, y) \in E$, then $(y, x) \in E$. If $(x, y) \in E$, then we say that xy , or equivalently yx , is an edge of G .

Definition 2.6. Let $D = (V, E)$ be a digraph and let $u, v \in V$. A *u-v walk* (in D), or a walk from u to v , is a finite sequence of vertices $u = w_0, w_1, \dots, w_{n-1}, w_n = v$ with the property that $(w_i, w_{i+1}) \in E$ for every $i \in [0, n-1]$. A *closed u-v walk* is a $u-v$ walk with the property that $u = v$.

Definition 2.7. Let $D = (V, E)$ be a digraph and let $u, v \in V$. A *u-v path* (in D), or a path from u to v , is a $u-v$ walk with distinct vertices, except possibly u and v . A *closed u-v path* is a $u-v$ path with the property that $u = v$.

Remark 2.8. We sometimes name a $u-v$ walk P by using the symbolism

$$P : u = w_0, w_1, \dots, w_{n-1}, w_n = v.$$

We say that $w_i \in P$ or that w_i is on P .

Remark 2.9. If $G = (V, E)$ is a graph and $u, v \in V$, then if there exists a u - v path in G , then there exists a v - u path in G .

Definition 2.10. Let $D = (V, E)$ be a digraph and let $u, v \in V$. We say that v is *reachable* (in D) from u if there exists a path from u to v (in D). If u is reachable from v and v is reachable from u , then we say that the pair (u, v) , or (v, u) , is mutually reachable (in D).

Remark 2.11. If G is a graph, then reachable and mutually reachable are equivalent notions.

Definition 2.12. Let $D = (V, E)$ be a digraph. We say that D is *strongly connected* if and only if every pair of vertices $(u, v) \in V \times V$ is mutually reachable. If D is a strongly connected graph, then we say that D is connected.

We now set up the correspondence between template configurations and directed graphs.

Definition 2.13. For every $x \in X$, let $E_x = \{ (y, x) : y \in R(x) \}$. The *digraph of R* , denoted $D(R)$, is the digraph $D(R) = (X, E)$ where $E = \bigcup_{x \in X} E_x$.

Definition 2.14. We say that R is *symmetric* if and only if for every $x, y \in X$, $x \in R(y)$ implies that $y \in R(x)$.

Theorem 2.15. $D(R)$ is a graph if and only if R is symmetric.

Proof.

Assume that $D(R) = (X, E)$ is a graph. Let $x, y \in X$ with $x \in R(y)$. By definition, $(x, y) \in E$. Since $D(R)$ is a graph this implies that $(y, x) \in E$. Thus, $y \in R(x)$ so R is symmetric.

Conversely assume that R is symmetric and that $(x, y) \in E$. Then $x \in R(y)$ and therefore, since R is symmetric, $y \in R(x)$. Thus, $(y, x) \in E$ which implies that $D(R)$ is a graph.

Q.E.D.

If R is symmetric, then we write $G(R)$ for the graph of R .

We now state Tchuente's theorem using the notation of image algebra.

Theorem (Tchuente). *Let $X \subset \mathbb{Z}^k$ be a finite set and let $R: X \rightarrow 2^X$ be a symmetric template configuration with the property that $x \in R(x)$ for every $x \in X$. Every template $T \in L_X$ has a local decomposition with respect to R if and only if $G(R)$ is connected.*

Tchuente's theorem is not as general as one might like. The assumption that the configuration R is symmetric restricts the type of multiprocessors being modeled to those in which the data can flow in both directions along any of the communication links. In many important cases, such as pipeline and parallel-pipeline computers, and systolic arrays, this assumption is not valid [26, 30, 42].

We will prove the following generalization of Tchuente's theorem:

Theorem 2.16. *Let $X \subset \mathbb{Z}^k$ be a finite set and let $R: X \rightarrow 2^X$ be a template configuration with the property that $x \in R(x)$ for every $x \in X$. Every template $T \in L_X$ has a local decomposition with respect to R if and only if $D(R)$ is strongly connected.*

Although the two theorems appear to be almost identical, it happens that, as is often true in graph theory, the case for directed graphs is significantly different than for graphs [4]. There are two differences between graphs and digraphs which emerge in trying to use a straightforward generalization of Tchuente's proof. One difference is that any permutation of a graph can be accomplished by a sequence of adjacent

transpositions. This is not necessarily true for a digraph (consider the directed cycle). Another difference is that on a connected graph there is always at least one vertex which can be removed, along with the edges incident to it, such that the resulting graph is still connected. This is not necessarily true of a digraph (consider again the directed cycle). Tchuente relies upon both of these facts for graphs.

We will prove Theorem 2.16 in stages. We first prove that strong connectivity is a necessary condition. We then establish a sequence of theorems which result in showing that strong connectivity is sufficient for permutation templates. Finally, we show how certain matrix decompositions can be used in conjunction with the results on permutations to deduce that strong connectivity is sufficient in general.

2.2. Necessary and Sufficient Conditions for the Existence of Local Decompositions

We now show that strong connectivity is necessary.

Theorem 2.17. *If every $T \in L_X$ has a local decomposition with respect to R , then $D(R)$ is strongly connected.*

Proof.

Assume by way of contradiction that $D(R)$ is not strongly connected but that every $T \in L_X$ has a local decomposition with respect to R . Let $x, y \in X$ such that x is not reachable from y . Let

$$C_1 \equiv \{ u \in X : x \text{ is reachable from } u \}$$

and

$$C_2 \equiv \{ v \in X : v \text{ is reachable from } y \}.$$

Note that $C_1 \cap C_2 = \emptyset$ since if $\mathbf{z} \in C_1 \cap C_2$, then \mathbf{z} is reachable from \mathbf{y} and \mathbf{x} is reachable from \mathbf{z} which implies that \mathbf{x} is reachable from \mathbf{y} .

Assume that T is a local template with respect to R . We show that $t_u(\mathbf{v}) = 0$ if $\mathbf{u} \in C_1$ and $\mathbf{v} \in C_2$. If this is not true, then there must be a $\mathbf{u} \in C_1$ and $\mathbf{v} \in C_2$ such that $t_u(\mathbf{v}) \neq 0$ which implies that $\mathbf{v} \in T(\mathbf{u})$. Since T is local with respect to R , $T(\mathbf{u}) \subset R(\mathbf{u})$ so $\mathbf{v} \in R(\mathbf{u})$ which implies that \mathbf{u} is reachable from \mathbf{v} . However, \mathbf{x} is reachable from \mathbf{u} so \mathbf{x} is reachable from \mathbf{v} . Thus, we arrive at the contradiction $\mathbf{v} \in C_1$.

We now construct a template which cannot have a local decomposition with respect to R . Let $T \in L_X$ be defined by

$$T(\mathbf{z}) \equiv \begin{cases} E(\mathbf{y}) & \text{if } \mathbf{z} = \mathbf{x} \\ 0 & \text{otherwise} \end{cases}.$$

By assumption there exists a decomposition $T = \bigoplus_{i=1}^k T_i$ which is local with respect to R .

Let $A \in C^X$ be defined by

$$A \equiv \{ (\mathbf{z}, a(\mathbf{z})) : a(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{z} = \mathbf{y} \\ 0 & \text{otherwise} \end{cases} \}.$$

The image $B = A \oplus T$ is given by

$$B \equiv \{ (\mathbf{z}, b(\mathbf{z})) : b(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{z} = \mathbf{x} \\ 0 & \text{otherwise} \end{cases} \}.$$

Let $j \in [1, k]$. We show that if $B_j = A \oplus (\bigoplus_{i=1}^j T_i)$, then $b_j(\mathbf{u}) = 0$ for every $\mathbf{u} \in C_1$.

Assume that this is not true for B_1 . Then there exists a $\mathbf{u} \in C_1$ such that $b_1(\mathbf{u}) =$

$\sum_{\mathbf{v} \in T_1(\mathbf{u})} t_u^{(1)}(\mathbf{v})a(\mathbf{v}) \neq 0$. Therefore, there must exist a $\mathbf{v} \in T_1(\mathbf{u})$ such that $t_u^{(1)}(\mathbf{v}) \neq 0$ and

$a(\mathbf{v}) \neq 0$ which implies that $\mathbf{v} = \mathbf{y}$ which in turn implies that $\mathbf{v} \in C_2$. However, $\mathbf{v} \in C_2$

means that $t_u^{(1)}(\mathbf{v}) = 0$ which is a contradiction. Hence, the claim must be true for B_1 .

Assume that for some $j \in [2, k]$ the claim is true for B_1, B_2, \dots, B_{j-1} and that it is not true for B_j . Then there exists a $\mathbf{u} \in C_1$ such that $b_j(\mathbf{u}) = \sum_{\mathbf{v} \in T_j(\mathbf{u})} t_u^{(j)}(\mathbf{v}) b_{j-1}(\mathbf{v}) \neq 0$. Therefore, there must exist a $\mathbf{v} \in T_j(\mathbf{u})$ such that $t_u^{(j)}(\mathbf{v}) \neq 0$ and $b_{j-1}(\mathbf{v}) \neq 0$. By the induction hypothesis, $\mathbf{v} \notin C_1$. The statement that $\mathbf{v} \in T_j(\mathbf{u})$ implies that \mathbf{u} is reachable from \mathbf{v} since T_j is local. Moreover, since $\mathbf{u} \in C_1$, \mathbf{x} is reachable from \mathbf{v} so $\mathbf{v} \in C_1$ which is a contradiction. Hence, the claim must be true. In particular, it must be true for $B = B_k = A \oplus T$. Therefore, $b(\mathbf{x}) = 0$ which is another contradiction and proves the theorem.

Q.E.D.

Definition 2.18. Let $T \in L_X$. We say that T is a *permutation template* if and only if M_T is a permutation matrix for some $\Psi : L_X \rightarrow M_n$.

Remark 2.19. If $\Psi_1(T)$ is a permutation matrix for some ordering on \mathbf{X} , then $\Psi_2(T)$ is also a permutation matrix for any other ordering on \mathbf{X} .

Remark 2.20. Recall that the set of all $n \times n$ permutation matrices is isomorphic to the symmetric group Σ_n . Thus, in particular, since every permutation can be factored into a product of transpositions, every permutation template can be factored into a product of templates corresponding to transpositions. We identify these templates in the next definition.

Definition 2.21. For every $\mathbf{x}, \mathbf{y} \in \mathbf{X}$, denote by $T_{\mathbf{x}\mathbf{y}} \in L_X$ the template defined by

$$T_{xy}(z) = \begin{cases} E(x) & \text{if } z = y \\ E(y) & \text{if } z = x \\ E(z) & \text{otherwise} \end{cases}.$$

T_{xy} is called the *exchange template* associated with (x, y) , or simply an exchange template. Note that T_{xy} is the permutation template corresponding to the transposition $\sigma \in \Sigma_X$ defined by $\sigma \equiv (x, y)$. If $A \in C^X$ and $B = A \oplus T$, then

$$b(z) = \begin{cases} a(x) & \text{if } z = y \\ a(y) & \text{if } z = x \\ a(z) & \text{otherwise} \end{cases}.$$

Lemma 2.22. *Let $x, y \in X$ with $x \neq y$ and let T_{xy} be the exchange template associated with (x, y) . Assume that there exists an x - y path*

$$P_1 : x = u_0, u_1, \dots, u_{k-1}, u_k = y$$

and a y - x path

$$P_2 : y = w_0, w_1, \dots, w_{j-1}, w_j = x$$

such that the closed x - x walk

$$P : x = u_0, u_1, \dots, u_{k-1}, u_k, w_1, \dots, w_{j-1} = x$$

is a closed x - x path in $D(R)$. The exchange template T_{xy} has a local decomposition with respect to R .

Proof.

The proof is by construction. Denote the closed path P by

$$P : x = v_0, v_1, \dots, v_{n-1}, v_n = x.$$

Assume that $y = v_j$ where $j \in [1, n-1]$ and denote $a_i \equiv a(v_i)$. Note that

$$T_{xy} = T_{v_0 v_1} \oplus T_{v_1 v_2} \oplus T_{v_2 v_3} \oplus \dots \oplus T_{v_{j-1} v_j} \oplus T_{v_j v_{j+1}} \oplus \dots \oplus T_{v_{i-1} v_i} \oplus T_{v_i v_1}.$$

We show how $T_{v_0 v_1}$ can be expressed as a product of local templates. It is interesting to note that the factors of the decomposition are not permutation templates. Figure 2.1 is helpful in understanding the templates which we are about to describe.

Define local templates $T_{1,1}, T_{1,2}, \dots, T_{1,n-1}$ by

$$T_{1,k} \equiv \begin{cases} E(v_k) + E(v_{k-1}) & \text{if } z = v_k \\ E(z) & \text{otherwise} \end{cases}$$

for $k \in [1, n-1]$. The image $B_1 = A \oplus (\bigoplus_{k=1}^{n-1} T_{1,k})$ is given by

$$b_1(z) = \begin{cases} \sum_{i=0}^k a_i & \text{if } z = v_k \text{ for } k \in [0, n-1] \\ a(z) & \text{otherwise} \end{cases}.$$

Define the local template $T_{1,n}$ by

$$T_{1,n} \equiv \begin{cases} E(v_{n-1}) - E(v_0) & \text{if } z = v_0 \\ E(v_k) - E(v_{k-1}) & \text{if } z = v_k \text{ for } k \in [2, n-1] \\ E(z) & \text{otherwise} \end{cases}.$$

The image $B_2 = B_1 \oplus T_{1,n}$ is given by

$$b_2(z) = \begin{cases} \sum_{i=1}^{n-1} a_i & \text{if } z = v_0 \\ a_0 + a_1 & \text{if } z = v_1 \\ b_1(z) & \text{if } z = v_k \text{ for } k \neq 0, 1 \\ a(z) & \text{otherwise} \end{cases}.$$

Define local templates $T_{1,n+1}, T_{1,n+2}, \dots, T_{1,2n-3}$ by

$$T_{1,n+k} \equiv \begin{cases} E(v_{k+2}) + E(v_{k+1}) & \text{if } z = v_{k+2} \\ E(z) & \text{otherwise} \end{cases}$$

for $k \in [1, n-3]$. The image $B_3 = B_2 \oplus (\bigoplus_{k=1}^{n-3} T_{1,n+k})$ is given by

$$b_3(\mathbf{z}) = \begin{cases} \sum_{i=2}^k a_i & \text{if } \mathbf{z} = \mathbf{v}_k \text{ for } k \in [2, n-1] \\ b_2(\mathbf{z}) & \text{if } \mathbf{z} = \mathbf{v}_0, \mathbf{v}_1 \\ a(\mathbf{z}) & \text{otherwise} \end{cases}.$$

Define the local template $T_{1,2n-2}$ by

$$T_{1,2n-2} \equiv \begin{cases} E(\mathbf{v}_0) - E(\mathbf{v}_{n-1}) & \text{if } \mathbf{z} = \mathbf{v}_0 \\ E(\mathbf{v}_k) - E(\mathbf{v}_{k-1}) & \text{if } \mathbf{z} = \mathbf{v}_k \text{ for } k \in [3, n-1] \\ E(\mathbf{z}) & \text{otherwise} \end{cases}.$$

The image $B_4 = B_3 \oplus T_{1,2n-2}$ is given by

$$b_4(\mathbf{z}) = \begin{cases} a_1 & \text{if } \mathbf{z} = \mathbf{v}_0 \\ a_0 + a_1 & \text{if } \mathbf{z} = \mathbf{v}_1 \\ a(\mathbf{z}) & \text{otherwise} \end{cases}.$$

Define the local template $T_{1,2n-1}$ by

$$T_{1,2n-1} \equiv \begin{cases} E(\mathbf{v}_1) - E(\mathbf{v}_0) & \text{if } \mathbf{z} = \mathbf{v}_1 \\ E(\mathbf{z}) & \text{otherwise} \end{cases}.$$

The image $B_5 = B_4 \oplus T_{1,2n-1}$ is given by

$$b_5(\mathbf{z}) = \begin{cases} a_1 & \text{if } \mathbf{z} = \mathbf{v}_0 \\ a_0 & \text{if } \mathbf{z} = \mathbf{v}_1 \\ a(\mathbf{z}) & \text{otherwise} \end{cases}.$$

Hence, $B_5 = A \oplus T_{\mathbf{v}_0, \mathbf{v}_1}$ which enables us to conclude that $T_{\mathbf{v}_0, \mathbf{v}_1} = \bigoplus_{i=1}^{2n-1} T_{1,i}$ is a local decomposition of $T_{\mathbf{v}_0, \mathbf{v}_1}$. Similarly, we can construct local decompositions $\{T_{k,h}\}_{h=1}^{2n-1}$ of $T_{\mathbf{v}_{k-1}, \mathbf{v}_k}$ for $k \in [2, j-1]$. We then have that

$$T = \left[\bigoplus_{k=1}^j T_{\mathbf{v}_{k-1}, \mathbf{v}_k} \right] \oplus \left[\bigoplus_{k=j-1}^1 T_{\mathbf{v}_{k-1}, \mathbf{v}_k} \right] = \left[\bigoplus_{k=1}^j \left(\bigoplus_{h=1}^{2n-1} T_{k,h} \right) \right] \oplus \left[\bigoplus_{k=j-1}^1 \left(\bigoplus_{h=1}^{2n-1} T_{k,h} \right) \right]$$

is a local decomposition of T .

Q.E.D.

A	B ₁	B ₂	B ₃	B ₄	B ₅
a ₀	a ₀	$\sum_{i=1}^{n-1} a_i$	$\sum_{i=1}^{n-1} a_i$	a ₁	a ₁
a ₁	a ₀ +a ₁	a ₀ +a ₁	a ₀ +a ₁	a ₀ +a ₁	a ₀
a ₂	$\sum_{i=0}^2 a_i$	a ₂	a ₂	a ₂	a ₂
a ₃	$\sum_{i=0}^3 a_i$	a ₃	a ₂ +a ₃	a ₃	a ₃
.
.
a _{n-2}	$\sum_{i=0}^{n-2} a_i$	a _{n-2}	$\sum_{i=2}^{n-2} a_i$	a _{n-2}	a _{n-2}
a _{n-1}	$\sum_{i=0}^{n-1} a_i$	a _{n-1}	$\sum_{i=2}^{n-1} a_i$	a _{n-1}	a _{n-1}

Figure 2.1. Stages in the local implementation of $T_{\mathbf{v}_0 \mathbf{v}_1}$.

Theorem 2.23. Let $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ and let $T_{\mathbf{x}\mathbf{y}}$ be the exchange template associated with (\mathbf{x}, \mathbf{y}) .

Assume that $D(R)$ is strongly connected. The exchange template $T_{\mathbf{x}\mathbf{y}}$ has a local decomposition with respect to R .

Proof.

Since $D(R)$ is strongly connected, every pair (\mathbf{x}, \mathbf{y}) is mutually reachable in $D(R)$.

Let

$$P_1 : \mathbf{x} = \mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{n-1}, \mathbf{u}_n = \mathbf{y}$$

$$P_2 : \mathbf{y} = \mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{m-1}, \mathbf{v}_m = \mathbf{x}.$$

be \mathbf{x} - \mathbf{y} and \mathbf{y} - \mathbf{x} paths respectively. P_2 and P_1 can have only finitely many vertices in common other than \mathbf{x} and \mathbf{y} . If there are none, then, by Lemma 2.22, we are done.

Otherwise, label the common vertices $u_{i_1}, u_{i_2}, \dots, u_{i_k}$ where $i_1 < i_2 < \dots < i_k$. For convenience we take $i_0 \equiv 0$ and $i_{k+1} \equiv n$. Define j_1, j_2, \dots, j_k by $v_{j_s} \equiv u_{i_s}$ for $s \in [1, k]$. By our choice of the u_{i_s} , the closed walks

$$W_0 : x = u_0, u_1, \dots, u_{i_1} = v_{j_1}, v_{j_1+1}, \dots, v_m = x$$

$$W_1 : u_{i_1}, u_{i_1+1}, \dots, u_{i_2} = v_{j_2}, v_{j_2+1}, \dots, v_{j_1}$$

$$\vdots$$

$$W_k : u_{i_k}, u_{i_k+1}, \dots, u_{i_{k+1}} = v_{j_{k+1}}, v_{j_{k+1}+1}, \dots, v_{j_k}$$

are all closed paths. Furthermore,

$$T_{x,y} = T_{x,u_{i_1}} \oplus T_{u_{i_1},u_{i_2}} \oplus \dots \oplus T_{u_{i_{k-1}},u_{i_k}} \oplus T_{u_{i_k},y} \oplus T_{u_{i_{k-1}},u_{i_k}} \oplus \dots \oplus T_{u_{i_1},u_{i_2}} \oplus T_{x,u_{i_1}}.$$

Since each of the W_i are closed paths, Lemma 2.22 implies that the templates $T_{u_{j_i},u_{j_i+1}}$ for $j \in [0, k-1]$ have local decompositions with respect to R . Hence, $T_{x,y}$ has a local decomposition with respect to R .

Q.E.D.

Corollary 2.24. Assume that $D(R)$ is strongly connected and let $T \in L_X$ be a permutation template. Then T has a local decomposition with respect to R .

Proof.

Every permutation template can be factored into a product of exchange templates. By Theorem 2.23, every exchange template has a local decomposition with respect to R .

Q.E.D.

Thus, strong connectivity of the template configuration is a necessary and sufficient condition for the existence of local decompositions of all permutation templates. We now

show that this is so in the general case.

Lemma 2.25. Assume that $|\mathbf{X}| = n > 1$, $D(R)$ is strongly connected, and that, for some $j \in [1, n-1]$, $T_{j1}, T_{j2} \in L_X$ such that relative to some ordering of \mathbf{X} ,

$$\Psi(T_{j1}) \equiv M_{j1} = \left[\begin{array}{c|c|c|c|c} & I_{j-1} & & 0 & \\ & & & & \\ & & & & \\ \hline 0 & & \lambda_j & & \lambda_{j+1} & \cdots & \lambda_n \\ \hline & & & & & & \\ 0 & & 0 & & & & I_{n-j} \end{array} \right]$$

and

$$\Psi(T_{j2}) \equiv M_{j2} = \left[\begin{array}{c|c|c|c|c} & I_{j-1} & & 0 & \\ & & & & \\ & & & & \\ \hline 0 & & \mu_j & & 0 & \cdots & 0 \\ \hline & & & & & & \\ 0 & & \mu_{j+1} & & & & I_{n-j} \\ & & & & & & \\ & & & & & & \mu_n \end{array} \right].$$

The templates T_{j1} and T_{j2} have local decompositions with respect to R .

Proof.

Assume that $\mathbf{X} = \{x_1, x_2, \dots, x_n\}$ is the ordering used to define $\Psi: L_X \rightarrow M_n$.

We consider T_{j1} first. Note that if $A \in C^X$, then

$$b(\mathbf{x}) = \begin{cases} a(\mathbf{x}) & \text{if } \mathbf{x} \neq \mathbf{x}_j \\ \sum_{k=j}^n \lambda_k a(\mathbf{x}_k) & \text{if } \mathbf{x} = \mathbf{x}_j \end{cases}.$$

This can be seen by inspection of the matrix M_{ji} and recalling the definition of Ψ and $\nu: C^X \rightarrow C^n$. Since $D(R)$ is strongly connected and $n > 1$, there exists $y \in X$ such that $y \neq x_j$ and $y \in R(x_j)$, say $y = x_m$. The rough idea of the proof is to first multiply $a(x_j)$ by λ_j and leave everything else fixed. Each $a(x_k)$ for $k \in [j+1, n]$ is then "moved" to the location x_m , $\lambda_k a(x_k)$ is added to the current value at the location x_j , and $a(x_k)$ is then moved back to location x_k . All these steps can be done locally.

We now define the templates of the local decomposition. Define $T_j \in L_X$ by

$$T_j(x) \equiv \begin{cases} E(x) & \text{if } x \neq x_j \\ \lambda_j E(x_j) & \text{if } x = x_j \end{cases}.$$

T_j is clearly local and if $B_1 = A \oplus T_j$, then

$$b_1(x) \equiv \begin{cases} a(x) & \text{if } x \neq x_j \\ \lambda_j a(x_j) & \text{if } x = x_j \end{cases}.$$

For each $i, k \in [1, n]$, let P_{ik} denote the exchange template corresponding to (x_i, x_k) .

Recall that $P_{ik} = P_{ki} = P_{ik}^{-1}$. For each $k \in [j+1, n]$ define $U_k, T_k \in L_X$ by

$$U_k(x) \equiv \begin{cases} E(x) & \text{if } x \neq x_j \\ E(x_j) + \lambda_k E(x_m) & \text{if } x = x_j \end{cases}.$$

and

$$T_k \equiv P_{km} \oplus U_k \oplus P_{mk}.$$

Since $x_m \in R(x_j)$, each U_k is local with respect to R . Furthermore, by Theorem 2.23, each P_{km} has a local decomposition with respect to R . Therefore, for every $k \in [j+1, n]$, T_k has a local decomposition with respect to R .

If $A \in C^X$ and $B = A \oplus T_k$ for some $k \in [j+1, n]$, then

$$b(x) \equiv \begin{cases} a(x) & \text{if } x \neq x_j \\ a(x_j) + \lambda_k a(x_k) & \text{if } x = x_j \end{cases}.$$

Hence, for every $A \in C^X$, if $B = A \oplus (\bigoplus_{k=j}^n T_k)$, then

$$b(\mathbf{x}) = \begin{cases} a(\mathbf{x}) & \text{if } \mathbf{x} \neq \mathbf{x}_j \\ \sum_{k=j}^n \lambda_k a(\mathbf{x}_k) & \text{if } \mathbf{x} = \mathbf{x}_j \end{cases}.$$

Thus, $T_{j1} = \bigoplus_{k=j}^n T_k$ which enables us to conclude that T_{j1} has a local decomposition with respect to R .

We now prove the lemma for T_{j2} . The concept is the same; there are some points which must be modified to account for the transposition. Note that if $A \in C^X$ and $B = A \oplus T_{j2}$, then

$$b(\mathbf{x}) = \begin{cases} a(\mathbf{x}) & \text{if } \mathbf{x} = \mathbf{x}_k \text{ for } k \in [1, j-1] \\ \mu_j a(\mathbf{x}_j) & \text{if } \mathbf{x} = \mathbf{x}_j \\ \mu_k a(\mathbf{x}_j) + a(\mathbf{x}_k) & \text{if } \mathbf{x} = \mathbf{x}_k \text{ for } k \in [j+1, n] \end{cases}.$$

Define the local template $S_{n+1} \in L_X$ by

$$S_{n+1}(\mathbf{x}) \equiv \begin{cases} E(\mathbf{x}) & \text{if } \mathbf{x} \neq \mathbf{x}_j \\ \mu_j E(\mathbf{x}_j) & \text{if } \mathbf{x} = \mathbf{x}_j \end{cases}.$$

Since $D(R)$ is strongly connected, for every $k \in [j+1, n]$ there exists an $m_k \in [1, n]$ such that $\mathbf{x}_{m_k} \in R(\mathbf{x}_k)$ and $\mathbf{x}_{m_k} \neq \mathbf{x}_k$. For every such k define $V_k, S_k \in L_X$ by

$$V_k(\mathbf{x}) \equiv \begin{cases} E(\mathbf{x}) & \text{if } \mathbf{x} \neq \mathbf{x}_k \\ E(\mathbf{x}_k) + \mu_k E(\mathbf{x}_{m_k}) & \text{if } \mathbf{x} = \mathbf{x}_k \end{cases}$$

and

$$S_k \equiv P_{jm_k} \oplus V_k \oplus P_{jm_k}$$

where P_{mj} denotes an exchange template. Each S_k for $k \in [j+1, n]$ has a local decomposition since each V_k is local and the other factors are exchange templates. Moreover, if

$A \in \mathbf{C}^X$ and $B = A \oplus S_k$ for some $k \in [j+1, n]$, then

$$b(x) = \begin{cases} a(x) & \text{if } x \neq x_k \\ a(x_k) + \mu_k a(x_j) & \text{if } x = x_k \end{cases}.$$

Thus, for every $A \in \mathbf{C}^X$, $A \oplus T_{j2} = A \oplus (\bigoplus_{k=j}^n S_{k+1})$ so $T_{j2} = \bigoplus_{k=j}^n S_{k+1}$ which enables us to conclude that T_{j2} has a local decomposition with respect to R .

Q.E.D.

Lemma 2.26. *Let $M \in M_n$ with $n > 1$. For every $j \in [1, n-1]$, there exists $n \times n$ permutation matrices P_j and Q_j , $n \times n$ matrices M_{j1} and M_{j2} , of the form given in Lemma 2.25, and a constant $c \in \mathbf{C}$ such that*

$$M = \left[\prod_{j=1}^{n-1} P_j M_{j1} \right] \left[\begin{array}{ccc|ccc} & & & & 0 & & \\ & & & & \cdot & & \\ & & & & \cdot & & \\ & & & & 0 & & \\ \hline 0 & \cdot & 0 & | & c & & \end{array} \right] \left[\prod_{j=1}^{n-1} M_{j2} Q_j \right].$$

Proof.

We make use of an observation made by Tchuente that there exists permutation matrices P and Q , constants $\lambda_1, \lambda_2, \dots, \lambda_n, \mu_1, \mu_2, \dots, \mu_n$, and an $n-1 \times n-1$ matrix C such that

$$M = P \left[\begin{array}{ccc|ccc} \lambda_1 & & & \lambda_2 & \cdot & \lambda_n \\ \hline 0 & & & & & \\ \cdot & & & & I_{n-1} & \\ 0 & & & & & \end{array} \right] \left[\begin{array}{ccc|ccc} 1 & & 0 & \cdot & 0 \\ \hline 0 & & & & \\ \cdot & & & C & \\ 0 & & & & \end{array} \right] \left[\begin{array}{ccc|ccc} \mu_1 & & 0 & \cdot & 0 \\ \hline \mu_2 & & & & \\ \cdot & & & & I_{n-1} \\ \mu_n & & & & \end{array} \right] Q.$$

The proof is by induction on n . If $n = 2$, then the statement of the lemma is identical to the observation made by Tchuente. Assume that $n \geq 3$ and that the theorem is

true for $2, 3, \dots, n-1$. By Tchuente's observation there exists permutation matrices P_1 and Q_1 , $n \times n$ matrices M_{11} and M_{12} of the form given in Lemma 2.25, and an $n-1 \times n-1$ matrix C such that

$$M = P_1 M_{11} \left[\begin{array}{c|ccc} 1 & & 0 & & 0 \\ \hline 0 & & & & \\ \cdot & & & C & \\ 0 & & & & \end{array} \right] M_{12} Q_1.$$

By the induction hypothesis, for every $j \in [1, n-2]$, there exists permutation matrices \hat{P}_j , \hat{Q}_j , $n \times n$ matrices \hat{M}_{j1} , \hat{M}_{j2} , and a constant $c \in F$ such that

$$C = \left[\prod_{j=1}^{n-2} \hat{P}_j \hat{M}_{j1} \right] \left[\begin{array}{c|ccc} & & 0 & \\ & I_{n-2} & & \cdot \\ & & & \cdot \\ & & & 0 \\ \hline 0 & & 0 & c \end{array} \right] \left[\prod_{j=1}^{n-2} \hat{M}_{j2} \hat{Q}_j \right].$$

Define

$$P_{j+1} \equiv \left[\begin{array}{c|ccc} 1 & & 0 & & 0 \\ \hline 0 & & & & \\ \cdot & & & \hat{P}_j & \\ 0 & & & & \end{array} \right]$$

and, for $j \in [1, n-2]$, define M_{j1} , M_{j2} , and Q_j in a similar fashion. Since multiplication of block diagonal matrices can be accomplished by multiplying corresponding blocks,

$$\left[\begin{array}{c|ccc} 1 & & 0 & & 0 \\ \hline 0 & & & & \\ \cdot & & & C & \\ 0 & & & & \end{array} \right] =$$

$$\begin{aligned}
&= \left[\begin{array}{c|ccc} 1 & & 0 & & 0 \\ \hline 0 & & & & \\ \hline & & \prod_{j=1}^{n-2} \hat{P}_j \hat{M}_{j1} & & \\ \hline 0 & & & & \end{array} \right] \left[\begin{array}{c|ccc} & & & 0 \\ \hline & I_{n-1} & & \cdot \\ & & & \cdot \\ & & & 0 \\ \hline 0 & & 0 & | & c \end{array} \right] \left[\begin{array}{c|ccc} 1 & & 0 & & 0 \\ \hline 0 & & & & \\ \hline & & \prod_{j=1}^{n-2} \hat{M}_{j2} \hat{Q}_j & & \\ \hline 0 & & & & \end{array} \right] = \\
&= \left[\begin{array}{c|ccc} & & & 0 \\ \hline & I_{n-1} & & \cdot \\ & & & \cdot \\ & & & 0 \\ \hline 0 & & 0 & | & c \end{array} \right] \left[\begin{array}{c} \prod_{j=2}^{n-1} P_j M_{j1} \\ \prod_{j=2}^{n-1} M_{j2} Q_j \end{array} \right].
\end{aligned}$$

Hence,

$$M = \left[\begin{array}{c|ccc} & & & 0 \\ \hline & I_{n-1} & & \cdot \\ & & & \cdot \\ & & & 0 \\ \hline 0 & & 0 & | & c \end{array} \right] \left[\begin{array}{c} \prod_{j=1}^{n-1} P_j M_{j1} \\ \prod_{j=1}^{n-1} M_{j2} Q_j \end{array} \right]$$

which proves the lemma.

Q.E.D.

We now show that strong connectivity is sufficient in general.

Theorem 2.27. *If $D(R)$ is strongly connected, then every template $T \in L_X$ has a local decomposition with respect to R .*

Proof.

Let $T \in L_X$. We can write

$$M_T = \left[\prod_{j=1}^{n-1} P_j M_{j1} \right] \left[\begin{array}{ccc|ccc} & & & & 0 & \\ & & & & \cdot & \\ & & & & \cdot & \\ & & & & 0 & \\ & & & & & \\ \hline 0 & \cdot & 0 & | & c & \end{array} \right] \left[\prod_{j=1}^{n-1} M_{j2} Q_j \right]$$

where the P_j and Q_j are permutation matrices and the M_{ji} are $n \times n$ matrices of the form given in Lemma 2.25. For every $j \in [1, n-1]$ and $i \in [1, 2]$, let $T_{ji} \equiv \Psi^{-1}(M_{ji})$, $U_j \equiv \Psi^{-1}(P_j)$, $V_j \equiv \Psi^{-1}(Q_j)$, and

$$S \equiv \Psi^{-1} \left(\left[\begin{array}{ccc|ccc} & & & & 0 & \\ & & & & \cdot & \\ & & & & \cdot & \\ & & & & 0 & \\ & & & & & \\ \hline 0 & \cdot & 0 & | & c & \end{array} \right] \right).$$

By Lemma 2.26,

$$T = \left(\bigoplus_{j=n-1}^1 V_j \oplus T_{j2} \right) \oplus S \oplus \left(\bigoplus_{j=n-1}^1 T_{j1} \oplus U_j \right).$$

Since the V_j and U_j are permutation templates, by Corollary 2.24, they have local decompositions. By Lemma 2.25, for every $j \in [1, n-1]$ and $i \in [1, 2]$, T_{ji} has a local decomposition. The template S is local since $\Psi(S)$ is diagonal. Hence, T has a local decomposition with respect to R .

Q.E.D.

This concludes the sequence of theorems used to prove Theorem 2.16.

We have shown how correspondences between directed graphs, template configurations, networks of processors, and matrices can be used to provide necessary and sufficient conditions for the existence of local decompositions of linear transforms. We did not address the problem of developing efficient algorithms for obtaining such

decompositions in this chapter. We remark in closing that factoring a linear transformation into a product of local linear transformations is not the only conceivable method of implementing one locally. As we shall see, it is a good method for obtaining local algorithms in some cases.

CHAPTER 3

TRANSLATION INVARIANT AND CIRCULANT TEMPLATES AND GENERALIZATIONS

In Chapter 3 we define and generalize translation invariant and circulant templates. These templates occur often in digital image processing. The chapter is divided into three sections. The first section consists mainly of basic definitions and properties, and descriptions of the matrices corresponding to translation invariant and circulant templates. As we mentioned in the introduction to Part I, these templates are used to implement convolutions and therefore are directly related to the discrete Fourier transform. We describe how these relationships are manifested in the setting of the image algebra. In the second section, we describe a family of relationships between circulant templates and polynomials. We show how the problem of finding local decompositions of circulant templates is equivalent to that of factoring multivariable polynomials. In the case of separable circulants, the problem reduces to the single variable case. In the third section, we generalize the notion of a circulant template by defining a class of templates which we call G-templates. G-templates are translation invariant with respect to Cayley networks, which are networks whose underlying graph is the group graph of some finite group G . Cayley networks are being studied as possible models for parallel computer architectures [53]. Thus G-templates have potential applications to parallel image processing. We show that the set of all G-templates is an algebra which is isomorphic to the group algebra of G over the complex numbers. We also show that the invertibility of G-templates is directly linked to the invertibility of the discrete Radon transform on finite groups [11].

3.1. Basic Definitions and Relationships

In this section we define translation invariant and circulant templates and describe some of their basic properties. We remark that many of the results of this section are essentially known in one form or another. They are new results in the sense that they are theorems about objects in the image algebra. One goal in developing an algebraic structure for digital image processing is to understand the relationship of the operations used to the existing theory. Therefore, it is useful to assemble these facts within the setting of the image algebra. The inclusion of these results also helps to keep this dissertation complete and accessible to researchers who may not be familiar with some of the existing theory.

In this chapter, we shall consider matrices and vectors to be indexed by sets of the form $\{0, 1, \dots, n-1\}$.

Definition 3.1. Let X be a finite subset of Z^k . We say that $T \in L_X$ is *translation invariant* if and only if for every translation $\phi : Z^k \rightarrow Z^k$, if $\phi(x), \phi(y) \in X$, then the equation $t_x(y) = t_{\phi(x)}(\phi(y))$ holds.

Theorem 3.2. If $T \in L_X$ is translation invariant, then the configuration function satisfies

$$(T(x) + z) \cap X = T(x+z) \cap (X + z)$$

for every $x, z \in Z^k$ such that $x, x+z \in X$.

Proof.

Assume that $x, z \in Z^k$ such that $x, x+z \in X$.

Let $\bar{y} \in (T(x) + z) \cap X$. Then $\bar{y} = y+z$ for some $y \in T(x)$. Let $\phi : Z^k \rightarrow Z^k$ be defined by $\phi(u) \equiv u+z$. Then $\phi(x), \phi(y) \in X$. Hence $t_{x+z}(y+z) = t_{\phi(x)}(\phi(y)) = t_x(y) \neq$

0 which implies that $\bar{y} = y+z \in T(x+z)$. Since $\bar{y} \in X$ it follows that $\bar{y} \in T(x+z) \cap (X+z)$.

Conversely, assume that $\bar{y} \in T(x+z) \cap (X+z)$. Since $\bar{y} \in (X+z)$, there exists a $y \in X$ such that $\bar{y} = y+z$. Since $\bar{y} \in T(x+z)$, $t_{x+z}(y+z) \neq 0$. Let $\phi: Z^k \rightarrow Z^k$ be defined by $\phi(u) \equiv u-z$. Then $t_{x+z}(y+z) = t_{\phi(x)}(\phi(y)) = t_x(y) \neq 0$ which implies that $y \in T(x)$ so $\bar{y} \in (T(x) + z)$. By definition $T(x+z) \subset X$ so we may conclude that $\bar{y} \in (T(x) + z) \cap X$.

Q.E.D.

Definition 3.3. Let $A = (a_{ij})$ be an $n \times n$ matrix. We say that A is a *Toeplitz matrix* if and only if for every $i, j \in [0, n-1]$ and $k \in Z$ such that $i+k, j+k \in [0, n-1]$, $a_{ij} = a_{i+k, j+k}$. A Toeplitz matrix is constant along the diagonals.

Definition 3.4. Let $A = (a_{ij})$ be an $mn \times mn$ matrix. We say that A is *block Toeplitz with Toeplitz blocks* if and only if

$$A = \begin{bmatrix} A_0 & A_1 & \cdot & \cdot & \cdot & A_{m-1} \\ A_{-1} & A_0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & A_1 \\ A_{-(m-1)} & A_{-(m-2)} & \cdot & \cdot & A_{-1} & A_0 \end{bmatrix}$$

where for every $i \in [-(m-1), m-1]$, A_i is a Toeplitz matrix.

For the rest of this section, let $X = \{ (i, j) : 0 \leq i \leq m-1, 0 \leq j \leq n-1 \}$ be an $m \times n$ array. The following results can also be formulated in a straightforward fashion for higher dimensional arrays. As we remarked at the end of Chapter 1, we assume that the matrix corresponding to a template is constructed using the row major lexicographic ordering.

Theorem 3.5. Let $T \in L_X$ be translation invariant and let $M_T = (\mu_{ij})$ be the matrix corresponding to T . Then M_T is block Toeplitz with Toeplitz blocks.

Proof.

Since M_T is an $mn \times mn$ matrix, we can write

$$M_T = \begin{bmatrix} A_{00} & A_{01} & \cdot & \cdot & A_{0,m-1} \\ A_{10} & A_{11} & \cdot & \cdot & A_{1,m-1} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ A_{m-1,0} & A_{m-1,1} & \cdot & \cdot & A_{m-1,m-1} \end{bmatrix}$$

where each A_{ij} is an $n \times n$ matrix. We first show that if $i, j \in [0, m-1]$, then A_{ij} is a Toeplitz matrix. Let $A_{ij} = (a_{st})$ where $s, t \in [0, n-1]$. Then $a_{st} = \mu_{in+s, jn+t} = t_{(i,s)}(j, t)$. Fix s and t and assume that $k \in \mathbb{Z}$ such that $s+k, t+k \in [0, n-1]$. Then $a_{s+k, t+k} = \mu_{in+s+k, jn+t+k} = t_{(i, s+k)}(j, t+k) = t_{(i,s)}(j, t) = a_{st}$ which implies that A_{ij} is Toeplitz.

We now show that M_T is block Toeplitz; that is, if $k \in \mathbb{Z}$ such that $i+k, j+k \in [0, m-1]$, then $A_{i+k, j+k} = A_{ij}$. Let $A_{i+k, j+k} = (b_{st})$ and fix $s, t \in [0, n-1]$. Note that $a_{st} = \mu_{in+s, jn+t} = t_{(i,s)}(j, t)$ and $b_{st} = \mu_{(i+k)n+s, (j+k)n+t} = t_{(i+k,s)}(j+k, t)$. Since $i+k, j+k \in [0, m-1]$ and T is translation invariant, we conclude that $a_{st} = b_{st}$.

Q.E.D.

Corollary 3.6. The inverse of a translation invariant template is not necessarily translation invariant.

Proof.

The inverse of a block Toeplitz matrix with Toeplitz blocks is not necessarily Toeplitz [6, 24].

Q.E.D.

We now define circulant templates.

Remark 3.7. If $\mathbf{x} = (x_1, x_2) \in \mathbb{Z}^2$ then we denote $(x_1(\bmod m), x_2(\bmod n))$ by $\mathbf{x}(\bmod (m, n))$.

Definition 3.8. We say that $\phi : \mathbf{X} \rightarrow \mathbf{X}$ is a *circulant translation* if and only if ϕ is of the form $\phi(\mathbf{x}) = (\mathbf{x} + \mathbf{h})(\bmod (m, n))$ for some $\mathbf{h} \in \mathbf{X}$.

Note that if $\mathbf{z} \in \mathbb{Z} \times \mathbb{Z}$ and $\mathbf{z} \notin \mathbf{X}$, then $\mathbf{z}(\bmod (m, n)) = \mathbf{y} \in \mathbf{X}$ and $(\mathbf{x} + \mathbf{z})(\bmod (m, n)) = (\mathbf{x} + \mathbf{y})(\bmod (m, n))$. Therefore, we shall sometimes define circulants in terms of elements $\mathbf{z} \notin \mathbf{X}$.

Theorem 3.9. Let $\Phi = \{ \phi : \phi \text{ is a circulant translation on } \mathbf{X} \}$. The set Φ equipped with the operation of composition is a group which is isomorphic to $\mathbb{Z}_m \times \mathbb{Z}_n$.

Proof.

Note that $\mathbf{X} = \mathbb{Z}_m \times \mathbb{Z}_n$ as a set and can obviously be made into a group isomorphic to $\mathbb{Z}_m \times \mathbb{Z}_n$ using addition mod (m, n) . For each $\mathbf{h} \in \mathbf{X}$ denote by $\phi_{\mathbf{h}}$ the circulant translation defined by $\phi_{\mathbf{h}}(\mathbf{x}) = (\mathbf{x} + \mathbf{h})(\bmod (m, n))$. Note that $\phi \in \Phi$ if and only if $\phi = \phi_{\mathbf{h}}$ for some $\mathbf{h} \in \mathbf{X}$. Define $\psi : \mathbf{X} \rightarrow \Phi$ by $\psi(\mathbf{h}) \equiv \phi_{\mathbf{h}}$. ψ is an isomorphism.

Q.E.D.

Definition 3.10. We say that $T \in L_{\mathbf{X}}$ is *circulant* if and only if for every circulant translation ϕ , the equation $t_{\mathbf{x}}(\mathbf{y}) = t_{\phi(\mathbf{x})}(\phi(\mathbf{y}))$ holds. We denote the set of all circulants on \mathbf{X} by $C_{\mathbf{X}}$.

Remark 3.11. A circulant template is well defined by defining it at one point. That is, if

$T \in C_X$ and $T(i,j) = A$, then $T(x,y) = \{ (u,v, a(\phi^{-1}(u,v))) \}$ where ϕ is the circulant translation with the property that $\phi(i,j) = (x,y)$.

Theorem 3.12. *If $T \in C_X$ is circulant, then T is translation invariant.*

Proof.

Let $x, y \in X$, $h \in Z^2$, and $\phi : Z^2 \rightarrow Z^2$ the translation of the plane by h . Assume that $\phi(x), \phi(y) \in X$. Then $(x+h) = (x+h)(\text{mod}(m,n))$ and $(y+h) = (y+h)(\text{mod}(m,n))$. Hence $t_{\phi(x)}(\phi(y)) = t_{(x+h)(\text{mod}(m,n))}((y+h)(\text{mod}(m,n))) = t_x(y)$.

Q.E.D.

Definition 3.13. Let $C = (c_{ij})$ be an $n \times n$ matrix. We say that C is a *circulant matrix* of order n if and only if for every $k \in Z$, $c_{ij} = c_{(i+k)(\text{mod } n), (j+k)(\text{mod } n)}$. Thus

$$C = \begin{bmatrix} c_0 & c_1 & \cdot & \cdot & c_{n-1} \\ c_{n-1} & c_0 & \cdot & \cdot & c_{n-2} \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ c_1 & c_2 & \cdot & \cdot & c_0 \end{bmatrix}.$$

We write $C = \text{circ}(c_0, c_1, \dots, c_{n-1})$.

Definition 3.14. Let B be an $mn \times mn$ matrix. We say that B is a *block circulant matrix* with *circulant blocks* of type (m,n) if and only if there exists circulant matrices C_0, C_1, \dots, C_{m-1} such that $B = \text{circ}(C_0, C_1, \dots, C_{m-1})$.

The next theorem shows that circulant templates deserve their name.

Theorem 3.15. *Let $T \in C_X$ be a circulant template. The matrix M_T is a block circulant matrix with circulant blocks.*

Proof.

Denote $M_T \equiv (\mu_{ij})$. Since T is translation invariant it follows that M_T is block Toeplitz with Toeplitz blocks. Let

$$M_T \equiv \begin{bmatrix} A_0 & A_1 & \cdot & \cdot & A_{m-1} \\ A_{-1} & A_0 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & A_1 \\ A_{-(m-1)} & A_{-(m-2)} & \cdot & A_{-1} & A_0 \end{bmatrix}.$$

Let $h \in [0, m-1]$. We show that A_h is a circulant matrix. Denote $A_h \equiv (a_{ij})$. Then $a_{ij} = \mu_{i, hn+j} = t_{(0,i)}(h, j)$. Let $k \in \mathbb{Z}$. Then $a_{(i+k)(\text{mod } n), (j+k)(\text{mod } n)} = \mu_{(i+k)(\text{mod } n), hn+((j+k)(\text{mod } n))} = t_{(0, (i+k)(\text{mod } n))}(h, (j+k)(\text{mod } n))$. Define the circulant translation $\phi : \mathbf{X} \rightarrow \mathbf{X}$ by $\phi(u, v) \equiv (u, (v+k)(\text{mod } n))$. Then $a_{ij} = t_{(0,i)}(h, j) = t_{\phi(0,i)}(\phi(h, j)) = t_{(0, (i+k)(\text{mod } n))}(h, (j+k)(\text{mod } n)) = a_{(i+k)(\text{mod } n), (j+k)(\text{mod } n)}$. Thus we may conclude that A_h is a circulant matrix.

We now show that M_T is block circulant, that is, if $h \equiv k(\text{mod } m)$, then $A_h = A_k$. There are two possibilities, $h = k$, which is trivial, and (say) $h > 0$ and $k < 0$. In the latter case we have $h-k = m$. Let $A_h \equiv (a_{ij})$ and $A_k \equiv (b_{ij})$. Then $a_{ij} = \mu_{i, hn+j} = t_{(0,i)}(h, j)$ and $b_{ij} = \mu_{i, kn+j} = t_{(-k,i)}(0, j)$. Define the circulant translation $\phi : \mathbf{X} \rightarrow \mathbf{X}$ by $\phi(u, v) \equiv ((u+h)(\text{mod } m), v)$. Then $b_{ij} = t_{(-k,i)}(0, j) = t_{\phi(-k,i)}(\phi(0, j)) = t_{(0,i)}(h, j) = a_{ij}$ which shows that $A_h = A_k$ and proves the theorem.

Q.E.D.

Corollary 3.16. $(C_X, \oplus, +)$ is a commutative ring with unity.

Proof.

This follows immediately from the facts that the set of all block circulant matrices with circulant blocks of type (m, n) forms a commutative ring with unity [10] and that

the mapping $\Psi : L_X \rightarrow M_{mn}$ is an isomorphism.

Q.E.D.

As mentioned previously, translation invariant and circulant templates are used to implement convolutions in the image algebra. In fact, if $T \in C_X$, then the mapping $A \rightarrow A \oplus T$ is the *circular convolution* of the two-dimensional sequences $a_{ij} = a(i,j)$ and $b_{ij} = t_{(0,0)((-i,-j)(\text{mod}(m,n)))}$. Translation invariant templates occur often in digital image processing. The circulant templates are closely related to the discrete Fourier transform and the theory of fast convolutions. There are various techniques available for approximating translation invariant computations by circular convolutions [1, 37]. The simplest techniques, which are adequate for small convolutions, are either to pad the array with zeroes or to ignore the boundary effects.

The definitions and theorems presented in this section help to provide a link between image algebra and matrix algebra associated with the discrete Fourier transform. We now describe this link.

Definition 3.17. Let A be an $m \times n$ matrix and B an $s \times t$ matrix both with entries in F . The *Kronecker, or tensor, product* of A and B , denoted $A \otimes B$, is the $ms \times nt$ matrix given by

$$A \otimes B = \begin{bmatrix} a_{0,0}B & \cdot & \cdot & a_{0,n-1}B \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{m-1,0}B & \cdot & \cdot & a_{m-1,n-1}B \end{bmatrix}$$

It is well known that $(A \otimes B)(C \otimes D) = AC \otimes BD$ provided that the matrices are all of the appropriate dimensions. Hence, $\left[\prod_{i=0}^k A_i \right] \otimes I_j = \prod_{i=0}^k (A_i \otimes I_j)$.

Let $i \equiv \sqrt{-1}$.

Definition 3.18. Define $K \in L_X$ by

$$K(u,v) \equiv \frac{1}{\sqrt{nm}} \exp[-2\pi i(uX/n + vY/m)]$$

where $X, Y \in F^X$ are defined by $X = \{ (x,y,z) : x = z \}$ and $Y = \{ (x,y,z) : y = z \}$. K is called the *two-dimensional DFT template*. The transform $A \rightarrow A \oplus K$ is the DFT of A . K is an invertible template with inverse given by

$$K^{-1}(u,v) = \frac{1}{\sqrt{nm}} \exp[2\pi i(uX/n + vY/m)] .$$

Let $\omega_n \equiv \exp[-2\pi i/n]$.

Definition 3.19. Denote by F_n the $n \times n$ matrix with ij -th entry equal to $n^{-1/2} \omega_n^{ij}$ for $j \in [0, n-1]$, that is, $F_n \equiv n^{-1/2} (\omega_n^{ij})$. F_n is called the *one-dimensional Fourier matrix* of order n . If $x \in \mathbb{C}^n$ then $F_n x$ is the one-dimensional discrete Fourier transform (1-dim DFT) of x .

Definition 3.20. Let X be an $m \times n$ array and let K be the Fourier template. Let $F_{m \times n} \equiv \Psi(K)$. The matrix $F_{m \times n}$ is called the *two-dimensional Fourier matrix* of order $m \times n$.

Note that $F_{m \times n} \neq F_{mn}$ in general. Note further that $F_{m \times n} = F_m \otimes F_n = (F_m \otimes I_n)(I_m \otimes F_n)$. This last observation is an equation which expresses the fact that the two dimensional DFT can be computed by first computing one dimensional DFT's along each row and then along each column of the result.

Let $P = \text{circ}(0, 1, 0, \dots, 0)$ be $n \times n$. If $C = \text{circ}(c_0, c_1, \dots, c_{n-1})$, then let $f_C(x) = c_0 + c_1 x + \dots + c_{n-1} x^{n-1}$. Then $C = f_C(P)$. Let $\Omega = \text{diag}(\omega_n^i)$, $i=0, 1, \dots, n-1$. Then, using the fact that

$$\sum_{i=0}^{n-1} \omega_n^{ik} = \begin{cases} n & \text{if } k \equiv 0 \pmod{n} \\ 0 & \text{else} \end{cases}$$

it can be seen that $P = F_n \Omega F_n^*$ where $*$ denotes the conjugate transpose. It follows that if C is circulant, then $C = F_n \Lambda F_n^*$ where $\Lambda = \text{diag}(f_C(\omega_n^i))$, $i=0,1,\dots,n-1$. This is the matrix formulation of the circular convolution theorem. Similarly, if B is a block circulant matrix with circulant blocks of type (m,n) , then $B = (F_m \otimes F_n) D (F_m \otimes F_n)^*$ where D is a diagonal matrix.

Thus, the circulant matrices are all simultaneously diagonalizable by the Fourier matrices. In the language of image algebra, we can say that if $T \in C_X$, then there exists a template $S \in L_X$ such that $S(x) \subset \{x\}$ for every $x \in X$ and $T = ((A \oplus K) \oplus S) \oplus K^{-1}$. Note that the computation $B \oplus S$ can be implemented using pointwise multiplication. This is how the fast Fourier transform (FFT) can be used to facilitate the computation of convolutions.

The Fourier template can be seen as inducing an isomorphism of C_X onto C^X . This is the image algebra version of the convolution theorem.

Theorem 3.21. *The mapping $\Phi : C_X \rightarrow C^X$ defined by $\Phi(S) \equiv S(0,0) \oplus K$ is a ring isomorphism of C_X onto C^X .*

Proof.

If $R, S \in C_X$ then, since the mapping $A \rightarrow A \oplus K$ is linear and addition of templates is defined pointwise, $\Phi(R) + \Phi(S) = \Phi(R+S)$. By definition of the discrete Fourier transform, $\Phi(E) = E(0,0) \oplus K = I$. To see that Φ is onto, let $B \in C^X$, let $A = B \oplus K^{-1}$, and define $S \in C_X$ by taking $S(0,0) \equiv A$ and extending by circularity. Then $\Phi(S) = B$. Suppose that $\Phi(R) = \Phi(S)$. Then $R(0,0) \oplus K = S(0,0) \oplus K$ which implies that $R(0,0)$

$= S(0,0)$ since K is invertible. Since R and S are determined by their values at one point, we may conclude that $R = S$ and therefore that Φ is 1-1.

Let $T = R \oplus S$ and, for every $i, j \in X$, define $\phi_{ij} : X \rightarrow X$ by $\phi_{ij}(x, y) = (x-i \pmod m, y-j \pmod n)$. Then

$$\begin{aligned} T(0,0) &= \{ (x,y, t_{(0,0)}(x,y) : t_{(0,0)}(x,y) = \sum_{(i,j) \in X} s_{(0,0)}(i,j) r_{(i,j)}(x,y) = \\ &= \sum_{(i,j) \in X} s_{(0,0)}(i,j) r_{\phi(i,j)}(\phi(x,y)) = \\ &= \sum_{(i,j) \in X} s_{(0,0)}(i,j) r_{(0,0)}((x-i) \pmod m, (y-j) \pmod n) \} . \end{aligned}$$

The last expression for $t_{(0,0)}(x,y)$ represents the gray values of $T(0,0)$ as the cyclic convolution of the two dimensional sequences $\{ s_{(0,0)}(i,j) \}_{(i,j) \in X}$ and $\{ r_{(0,0)}(i,j) \}_{(i,j) \in X}$. By the convolution theorem, $T(0,0) \oplus K = (R(0,0) \oplus K) * (S(0,0) \oplus K)$ which implies that $\Phi(R \oplus S) = \Phi(R) * \Phi(S)$.

Q.E.D.

One of the useful properties of the discrete Fourier transform is that it converts convolutions into pointwise multiplications, a fact which is expressed in Theorem 3.21. Because of this, local algorithms for computing convolutions can be derived by deriving local algorithms for computing discrete Fourier transforms. A similar situation exists for other invertible linear transformations.

Definition 3.22. We say that $T \in L_X$ is a *diagonal template* if, relative to some ordering on X , M_T is a diagonal matrix. By Theorem 1.12, if M_T is diagonal relative to some ordering on X then for every ordering on X , $\Psi(T)$ is a diagonal matrix.

With each diagonal template T we associate the image A_T defined by $A_T \equiv \sum_{x \in X} T(x)$. Note that if $A_T = \{ (x, a(x)) \}$, then $a(x) = t_x(x)$. Moreover, $A \oplus T =$

$A \star A_T$.

Theorem 3.23. Assume that $T \in L_X$ is invertible. Define

$$L_X(T) \equiv \{ T^{-1} \oplus D \oplus T : D \text{ is a diagonal template} \}.$$

$L_X(T)$ is a commutative ring which is isomorphic to C^X .

Proof.

The set $L_X(T)$ is a commutative ring since $\Psi(L_X(T)) = \{ M_T \Delta M_T^{-1} : \Delta \text{ is a diagonal matrix} \}$ is a commutative ring. Define $\Phi : L_X(T) \rightarrow C^X$ by $\Phi(S) \equiv A_D$ where $D \equiv T \oplus S \oplus T^{-1}$ for $S \in L_X(T)$. The verification that Φ is an isomorphism is routine and we omit it.

Q.E.D.

Thus, if methods can be found for implementing invertible linear transforms locally, then those methods can be used to implement a larger class of linear transforms locally.

We have defined translation invariant and circulant templates and outlined their relationships to matrix algebra associated with the discrete Fourier transform. We now describe some relationships between circulant templates and polynomials.

3.2. Circulant Templates and Polynomials

In this section, we describe a family of relationships between circulant templates and quotient rings of polynomial rings. We show that the problem of finding decompositions of circulant templates is equivalent to factoring multivariable polynomials. We show that for separable circulant templates the problem reduces to the single variable case and is therefore equivalent to finding roots of polynomials in one variable. We show

how, by the use of shifts and the fundamental theorem of algebra, minimal local decompositions of separable circulant templates can be obtained. We develop the results in the two-dimensional setting. As was true in the previous section, they extend easily to the higher dimensional cases.

Throughout this section, $\mathbf{x}, \mathbf{y}, \mathbf{z}$ will denote elements of the $m \times n$ array \mathbf{X} , $0 \equiv (0,0)$, x and y will denote indeterminates, and $\mathbb{C}[x,y]$ the ring of polynomials in two variables with coefficients in \mathbb{C} . We let $\mathbb{C}[x,y]/(x^m-1, y^n-1)$ denote the quotient ring of polynomials mod x^m-1 and y^n-1 . We consider the elements of this ring to be polynomials on which multiplication is performed by replacing x^m and y^n with 1 wherever they appear rather than as cosets. We also denote by V the template configuration defined by $V(i,j) = \{ (i,j), (i+1 \pmod m, j), (i-1 \pmod m, j), (i, j+1 \pmod n), (i, j-1 \pmod n) \}$. V is called the *von Neumann* configuration. In this section, local shall mean local with respect to this particular configuration.

For every $\mathbf{z} \in \mathbf{X}$ we define a mapping $\Gamma_{\mathbf{z}} : \mathbb{C}_{\mathbf{X}} \rightarrow \mathbb{C}[x,y]/(x^m-1, y^n-1)$ by

$$\Gamma_{\mathbf{z}}(T) \equiv p_t(x,y,\mathbf{z}) \equiv \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} t_{\mathbf{z}}(i,j) x^i y^j.$$

If T is a circulant template then $\Gamma_{\mathbf{z}}$ is called a *polynomial representative* of T .

Example. Let R be the circulant template depicted in figure 3.1. Then

$$\Gamma_{(0,0)}(R) = 4 + 6y + 2y^{n-1} + 6x + 9xy + 3xy^{n-1} + 2x^{m-1} + 3x^{m-1}y + x^{m-1}y^{n-1}$$

and

$$\Gamma_{(1,1)}(R) = 1 + 2y + 3y^2 + 2x + 4xy + 6xy^2 + 3x^2 + 6x^2y + 9x^2y^2$$

$R =$

1	2	3
2	4	6
3	6	9

Figure 3.1. A circulant template R .

Properties.

1.) If T is a circulant template and $\Gamma_{(0,0)}(T) = x-a$ or $y-a$ for some a , then T is local.

2.) If T is a circulant template and $\Gamma_{(1,0)}(T) = a_0 + a_1x + a_2x^2$ or $\Gamma_{(0,1)}(T) = a_0 + a_1y + a_2y^2$, then T is local.

3.) If T is a circulant template and $\Gamma_{(0,0)}(T) = x^i y^j$, then the circulant transform $A \rightarrow A \oplus T$ simply circularly shifts all the gray levels i units vertically and j units horizontally.

Theorem 3.24. *If S and T are circulant templates, then $\Gamma_0(S \oplus T) = \Gamma_0(S)\Gamma_0(T)$.*

Proof.

For any $\mathbf{z} \in \mathbf{X}$ denote by $\phi_{\mathbf{z}}$ the circulant translation defined by $\phi_{\mathbf{z}}(\mathbf{x}) = (\mathbf{x} - \mathbf{z}) \bmod(m, n)$. Note that $\phi_{\mathbf{z}}(\mathbf{z}) = \mathbf{0}$. Recall that, if $R = S \oplus T$, then $R(0) = \{ (y, r_0(y)) : r_0(y) = \sum_{\mathbf{z} \in T(0)} t_0(\mathbf{z}) s_{\mathbf{z}}(y) \}$. Since S is a circulant $s_{\mathbf{z}}(y) = s_0(\phi_{\mathbf{z}}(y)) = s_0((y - \mathbf{z}) \bmod(m, n))$. But by definition of polynomial multiplication (in $C[x, y]/(x^m - 1, y^n - 1)$) the numbers $\sum_{\mathbf{z} \in T(0)} t_0(\mathbf{z}) s_0((y - \mathbf{z}) \bmod(m, n))$ are precisely the

coefficients of the polynomial product $p_s(x,y,0)p_t(x,y,0)$.

Q.E.D.

Theorem 3.25. For every $z = (j,k) \in X$, Γ_z is 1-1 and onto. Moreover, if $S, T \in C_X$, then

$$1.) \Gamma_z(S) + \Gamma_z(T) = \Gamma_z(S+T).$$

$$2.) \Gamma_z(T) = x^j y^k \Gamma_0(T).$$

$$3.) \Gamma_z(S \oplus T) x^j y^k = \Gamma_z(S) \Gamma_z(T).$$

Proof:

Let $z = (j,k) \in X$. The mapping Γ_z is clearly onto and is 1-1 since C_X contains only circulant templates with minimal configuration. Equation 1.) follows from the fact that addition of templates is defined pointwise.

To prove 2.) let ϕ be the circulant translation defined by $\phi(x) = (x+z) \bmod(m,n)$. Note that $\phi(0) = z$. Let T be an arbitrary circulant template. Then

$$\begin{aligned} x^j y^k p_t(x,y,0) &= x^j x^k \sum_{i=0}^{m-1} \sum_{h=0}^{n-1} t_0(i,h) x^i y^h = \\ &= \sum_{i=0}^{m-1} \sum_{h=0}^{n-1} t_z((i+j) \bmod m, (h+k) \bmod n) x^{i+j} y^{h+k} = \\ &= \sum_{i=0}^{m-1} \sum_{h=0}^{n-1} t_z(i,h) x^i y^h = p_t(x,y,z) \end{aligned}$$

since $x^m \equiv y^n \equiv 1$ and T is circulant. This enables us to deduce that $\Gamma_z(T) = x^j y^k \Gamma_0(T)$ or $x^{m-j} y^{n-k} \Gamma_z(T) = \Gamma_0(T)$.

To prove 3.), observe that, since $\Gamma_0(S \oplus T) = \Gamma_0(S) \Gamma_0(T)$, we have that

$$\begin{aligned}\Gamma_z(S \oplus T) &= x^j y^k \Gamma_0(S) \Gamma_0(T) = \\ &= x^j y^k \left[x^{m-j} y^{n-k} \Gamma_z(S) x^{m-j} y^{n-k} \Gamma_z(T) \right] = x^{m-j} y^{n-k} \Gamma_z(S) \Gamma_z(T).\end{aligned}$$

The desired result now follows by multiplying the last equation by $x^j y^k$.

Q.E.D.

Example. Let R be the circulant template depicted in figure 3.1. Note that $\Gamma_{(1,1)}(R) = xy\Gamma_{(0,0)}(R)$. Let S and T be the circulant templates depicted in figure 3.2. Then $S \oplus T = R$, $\Gamma_{(1,1)}(S) = y + 2xy + 3x^2y$, $\Gamma_{(1,1)}(T) = x + 2xy + 3xy^2$. A simple polynomial multiplication shows that $xy\Gamma_{(1,1)}(S \oplus T) = xy\Gamma_{(1,1)}(R) = \Gamma_{(1,1)}(S)\Gamma_{(1,1)}(T)$.

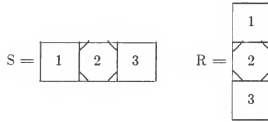


Figure 3.2. Circulant templates S and T .

Corollary 3.26. Γ_0 is an isomorphism.

Corollary 3.27. Let $\mathbf{z}_1 = (i, j)$, $\mathbf{z}_2 = (s, t) \in \mathbf{X}$ and $T \in C_X$. If $p_t(x, y, \mathbf{z}_1) = p_1(x, y)p_2(x, y)$, then $p_t(x, y, \mathbf{z}_2) = q_1(x, y)q_2(x, y)$ where $q_1(x, y) = x^{s-i}y^{t-j}p_1(x, y)$ and $q_2(x, y) = p_2(x, y)$.

Corollary 3.28. If $\mathbf{z} = (i, j) \in \mathbf{X}$, $T \in C_X$, and $p_t(x, y, \mathbf{z}) = p_1(x, y)p_2(x, y)$, then $T =$

$T_1 \oplus T_2 \oplus S$, where $T_1 = \Gamma_z^{-1}(p_1(x,y))$, $T_2 = \Gamma_z^{-1}(p_2(x,y))$, and S represents a circular shift of $n-j$ units horizontally and $m-i$ units vertically. Moreover, since S represents a circular shift, it can be replaced by a template \hat{S} which represents a circular shift j units horizontally and i vertically.

The Γ_z 's constitute a class of mappings between image and polynomial algebra which are almost isomorphisms. Since they differ from isomorphisms only by shifts, they can be used in the same way. Clearly if any of the polynomial representatives of a template can be factored, then the template can be factored correspondingly. Thus, the template decomposition problem is equivalent to the problem of factoring multivariable polynomials. It is interesting to observe that there is research being done in the area of using computer programs to determine exact factorizations of multivariable polynomials over the rationals [25, 34]. We now examine some specific consequences of these theorems. We will use them to show how any separable circulant template can be implemented locally with respect to the von Neumann configuration and give upper bounds on the number of parallel steps required. Since the von Neumann restriction on an $m \times n$ array simulates mesh-connected arrays of processors, these methods can be used on such machines.

Definition 3.29. If $f(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} x^i y^j$, then

$$\deg(f(x,y)) \equiv \max \{ i+j : a_{ij} \neq 0 \}.$$

$$\deg_x(f(x,y)) \equiv \max \{ i : a_{ij} \neq 0 \}.$$

$$\deg_y(f(x,y)) \equiv \max \{ j : a_{ij} \neq 0 \}.$$

Definition 3.30. If T is a circulant template and $\deg(p_i(x,y,z)) \leq \deg(p_i(x,y,w))$ for every

$w \in X$, then we say that z is a *minimal point* for T .

Definition 3.31. Let T be a circulant template. We say that T is *separable* if there exist polynomials $f(x)$ and $g(y)$ such that $p_t(x,y,0) = f(x)g(y)$. By the preceding corollaries if T is separable, then for every $z \in X$ there exists polynomials $f(x)$ and $g(y)$ such that $p_t(x,y,z) = f(x)g(y)$.

The following theorem yields a systematic method for computing any separable circulant transform locally with respect to a four-connected processor array and gives an upper bound on the number of parallel steps required.

Theorem 3.32. Let T be a separable, circulant template and let $z = (s,t)$ be a minimal point for T . Let $k = \deg_x(p_t(x,y,z))$ and $j = \deg_y(p_t(x,y,z))$. Let $v = \min(s,m-s)$ and $h = \min(t,n-t)$. The circulant transform $A \rightarrow A \oplus T$ can be computed in at most $v+h+k+j+1$ local, parallel steps. Specifically, $v+h$ of these steps consist of vertical or horizontal circular shifts of the entire array by one location, $k+j$ of the steps consist of at most one addition and one multiplication (possibly complex) per pixel, and one of the steps consists of at most one multiplication per pixel.

Proof.

Since T is separable, there exists $f(x)$ and $g(y)$ such that $p_t(x,y,z) = f(x)g(y)$. By assumption, $\deg(f(x)) = k$ and $\deg(g(y)) = j$. By the fundamental theorem of algebra, $f(x) = a(x-q_1)(x-q_2)\dots(x-q_k)$ and $g(y) = b(y-r_1)(y-r_2)\dots(y-r_j)$ where $a, b, q_1, \dots, q_k, r_1, \dots, r_j$ are (possibly) complex numbers. Choose circulant templates $Q, R, Q_1, \dots, Q_k, R_1, \dots, R_j$ such that $\Gamma_0(Q) = f(x)$, $\Gamma_0(R) = g(y)$, $\Gamma_0(Q_i) = x - q_i$, and $\Gamma_0(R_i) = y - r_i$. By Property 1.), the Q_i and R_i are local. By construction, $\Gamma_z(T) = \Gamma_0(Q)\Gamma_0(R)$. Furthermore, by Theorem 3.24, $\Gamma_0(Q) = a(\prod_{i=1}^k \Gamma_0(Q_i))$ which implies that $Q = a(\bigoplus_{i=1}^k Q_i)$. Similarly, $R = b(\bigoplus_{i=1}^j R_i)$.

Hence, by Corollary 3.28, $T = ab \left[\left(\bigoplus_{i=1}^k Q_i \right) \oplus \left(\bigoplus_{i=1}^j R_i \right) \oplus S \right]$. This last equation expresses T as a product of local templates. The number of steps required to implement the circulant transform $A \rightarrow A \oplus T$ is $k+j+1$ multiplication steps plus the number of shifts required which is $v+h$.

Q.E.D.

The templates are shown in Figure 3.3. The slashes indicate the location of the center pixel.

$$Q_i = \begin{array}{|c|} \hline \begin{array}{c} \diagup \quad \diagdown \\ -q_i \end{array} \\ \hline 1 \\ \hline \end{array} \qquad R_i = \begin{array}{|c|c|} \hline \begin{array}{c} \diagup \quad \diagdown \\ -r_i \end{array} & 1 \\ \hline \end{array}$$

Figure 3.3. Templates used in Theorem 3.34

As an example suppose \mathbf{X} is a 512×512 array and T is a 30×30 separable template. Then $\mathbf{z} = (14, 14)$ and $v = h = 14$. Hence, if $A \in K^{\mathbf{X}}$, then $A \rightarrow A \oplus T$ can be computed locally in parallel with 60 parallel steps consisting of at most one multiplication and addition per pixel, 1 step consisting of one multiplication per point, and a total of 28 unit horizontal or vertical circular shifts. Since the computation in its original form required 900 multiplications per point, it is clear that template decompositions can be used to derive algorithms which are more efficient with respect to the number of arith-

metric operations as well as parallel. Nontrivial examples of such templates are the discretizations of the Marr-Hildreth edge operators [66].

The next theorem also yields a systematic method for computing separable circulant transforms locally and avoids complex multiplications.

Theorem 3.33. *Assume that T is a separable circulant template with real weights and that, for every $\mathbf{z} \in \mathbf{X}$, $p_i(x, y, \mathbf{z}) = \Gamma_{\mathbf{z}}(T) = f_{\mathbf{z}}(x)$ is a polynomial in x only. Let $(p, 0)$ be a minimal point for T .*

1.) *Assume that $\deg(f_{(p,0)}(x)) = 2k$ for some $k \geq 1$ and let $v = \min(m - |k - p|, |k - p|)$. The circulant transform $A \rightarrow A \oplus T$ can be computed in $k + v + 1$ local, parallel steps. Specifically, v of these steps consist of vertical, circular shifts of the entire array by one location, k of these steps consist of at most 2 real multiplications and additions per pixel, and 1 step consists of at most 1 real multiplication per pixel.*

2.) *Assume that $\deg(f_{(p,0)}(x)) = 2k + 1$ for some $k \geq 1$. The conclusion of 1.) holds with k replaced by $k + 1$.*

Proof.

1.) We can write $\Gamma_{(p,0)}(T) = c \prod_{i=1}^k q_i(x)$ where the $q_i(x)$ are monic, quadratic polynomials with real coefficients and c is a real number. For each $i \in [1, k]$ define $Q_i \in L_X$ by $Q_i \equiv \Gamma_{(1,0)}^{-1}(q_i(x))$. By Property 2.), the Q_i are local templates. Furthermore,

$$\Gamma_0(T) = x^{-p} \Gamma_{(p,0)}(T) = cx^{-p} \prod_{i=1}^k \Gamma_{(1,0)}(Q_i) = cx^{k-p} \prod_{i=1}^k \Gamma_0(Q_i)$$

Hence, $T = c(\bigoplus_{i=1}^k Q_i) \oplus S$ where S is a shift template.

If $k > p$, then we can take S to be a shift of $v_1 = \min(k-p, m-(k-p))$ units. If $k \leq p$, then we can take S to be a shift of $v_2 = \min(p-k, m-(p-k))$ units. In any case, the shift can be executed in at most v steps.

2.) The proof is almost identical. We write $\Gamma_{(p,0)}(T) = c \left[\prod_{i=1}^k q_i(x) \right] p(x)$ where the $q_i(x)$ are monic, quadratic polynomials with real coefficients, $p(x)$ is a monic linear polynomial with real coefficients, and c is a real number. We take P to be the local template $P = \Gamma_0^{-1}(p(x))$. Taking the Q_i to be as in 1.) yields

$$\Gamma_0(T) = cx^{-p} \left[\prod_{i=1}^k \Gamma_{(1,0)}(Q_i) \right] \Gamma_0(P) = cx^{k-p} \left[\prod_{i=1}^k \Gamma_0(Q_i) \right] \Gamma_0(P).$$

The rest of the proof is identical to the latter part of the proof of part 1.).

Q.E.D.

Clearly an analogous theorem holds for circulant templates T with the property that for every $\mathbf{z} \in \mathbf{X}$, $\Gamma_{\mathbf{z}}(T) = g_{\mathbf{z}}(y)$ are polynomials in y only.

Let $\lceil x \rceil$ denote the smallest integer greater than or equal to x .

Corollary 3.34. Assume that T is a separable circulant template and let (p,q) be a minimal

point for T . Let $k = \left\lceil \deg_x \left(\frac{P_t(x, y, (p, q))}{2} \right) \right\rceil$ and $j = \left\lceil \deg_y \left(\frac{P_t(x, y, (p, q))}{2} \right) \right\rceil$. Let $v = \min(m - \lceil k - p \rceil, \lceil k - p \rceil)$ and $h = \min(n - \lceil j - q \rceil, \lceil j - q \rceil)$. The circulant transform $A \rightarrow A \oplus T$ can be computed in $k+j+v+h+1$ local, parallel steps. Specifically, $v+h$ of these steps consist of vertical or horizontal shifts of the entire array by one location, $k+j$ of these steps consist of at most 2 real multiplications and additions per point, and one of the steps consists of at most one real multiplication per point.

The templates corresponding to the quadratic polynomials used in this technique are

depicted in Figure 3.4.

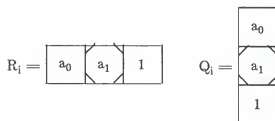


Figure 3.4. Quadratic templates used in Corollary 3.34.

Remark 3.35. The latter technique will generally be preferable to the former since one complex multiplication takes at least three real multiplications and five real additions or four real multiplications and two real additions [6].

We have described a family of relationships between circulant templates and polynomial algebra. We used these relationships to show how local decompositions of separable circulant templates can be obtained by factoring or finding roots of polynomials in a single variable. Finding roots of polynomials can be a numerically unstable procedure [38]. In recent years, research has been done on developing computer programs to factor polynomials exactly [25, 34]. This work could be applicable to the development of parallel algorithms for computing convolutions.

3.3. G-templates

In this section we generalize the notion of circulant templates by defining G-templates. A G-template will be defined as a template which is translation invariant

with respect to a digraph induced by a template configuration which admits of a group structure. In the case that the configuration is the von Neumann configuration, the group will turn out to be $\mathbf{Z}_m \times \mathbf{Z}_n$ and the set of all G-templates will be the algebra of all circulant templates. Similarly, we will show that the set of all G-templates for other configurations is isomorphic to the group algebra over \mathbf{C} of the group corresponding to the template configuration. We will also show that the invertibility of a G-template is directly related to the invertibility of the discrete Radon transform. We conclude this section with a discussion of the possible applications of G-templates to parallel processing.

Throughout this section, let $\mathbf{X} \subset \mathbf{Z}^k$ be a finite set, R a template configuration on \mathbf{X} , G a finite group (written multiplicatively), and assume that images have values in \mathbf{C} .

Definition 3.36. Let $\Delta = \{g_1, g_2, \dots, g_k\}$ be a set of generators of G . The *Cayley color graph*, or *group graph*, of G with respect to Δ is the digraph $D_\Delta(G) = (V, E)$ where $V = G$ and $(x, y) \in E$ if and only if there exists $g_i \in \Delta$ such that $xg_i = y$. If $(x, y) \in E$ and $xg_i = y$, then we say that the arc (x, y) is *colored* g_i , or has the color g_i .

Definition 3.37. An *automorphism* of a digraph is a permutation σ on $V(D)$ with the property that $(u, v) \in E(D)$ if and only if $(\sigma(u), \sigma(v)) \in E(D)$. A *color-preserving automorphism* of a Cayley color graph, $D_\Delta(G)$, is an automorphism, σ , of $D_\Delta(G)$ with the property that for every $x, y \in V(D_\Delta(G))$ the arcs (x, y) and $(\sigma(x), \sigma(y))$ have the same color.

Fact 3.38. The set of color preserving automorphisms of $D_\Delta(G)$ is a group under composition which is isomorphic to G . An isomorphism is the mapping $g \rightarrow \sigma_g$ where σ_g is the automorphism of $D_\Delta(G)$ defined by $\sigma_g(h) \equiv gh$. Note that this implies that the group of color preserving automorphisms is the same for every group graph constructed using

generating sets of G [4].

Definition 3.39. Let D_1 and D_2 be digraphs. We say that D_1 is isomorphic to D_2 if and only if there exists a 1-1, onto mapping, $\alpha: V(D_1) \rightarrow V(D_2)$, with the property that $(x, y) \in E(D_1)$ if and only if $(\alpha(x), \alpha(y)) \in E(D_2)$.

Definition 3.40. We say that the pair (X, R) *simulates* the group G if and only if $D(R)$ is isomorphic to $D_\Delta(G)$ as digraphs for some generating set Δ . We assign to each arc in $D(R)$ the same color as the associated arc in $D_\Delta(G)$.

Theorem 3.41. If (X, R) simulates G then a binary operation, \circ , can be defined on X which makes it a group isomorphic to G .

Proof.

Since (X, R) simulates G there exists a generating set, Δ , of G , and a 1-1, onto mapping $\alpha: V(D(R)) \rightarrow V(D_\Delta(G))$. By definition of the respective digraphs, $V(D(R)) = X$ and $V(D_\Delta(G)) = G$ which implies that $\alpha: X \rightarrow G$. Define \circ by $x \circ y \equiv \alpha^{-1}(\alpha(x)\alpha(y))$. Checking to see that X is a group under \circ is a routine matter and we omit it.

Q.E.D.

Remark 3.42. Henceforth, if (X, R) simulates G , then we identify X with G whenever it is convenient. If we order $X = \{x_0, x_1, \dots, x_{n-1}\}$, then we assume that $x_0 \equiv e$ is the identity element of G . If $\phi: G \rightarrow G$, then we can define a map $\phi^*: X \rightarrow X$ by $\phi^*(x) \equiv \alpha^{-1}(\phi(\alpha(x)))$. In such a case we write ϕ for either map.

Definition 3.43. Assume that (X, R) simulates G . We say that $T \in L_X$ is a G -template if

and only if for every color-preserving automorphism $\phi : \mathbf{X} \rightarrow \mathbf{X}$, the equation $t_{\phi(\mathbf{x})}(\phi(\mathbf{y})) = t_{\mathbf{x}}(\mathbf{y})$ holds. We denote by $G_{\mathbf{X}}$ the set of all G -templates.

Remark 3.44. The set of G -templates is, in a sense, the set of all templates which are translation invariant with respect to the group graph $D_{\Delta}(G)$ or $D(R)$ since the color-preserving automorphisms are essentially translations within the group.

We now show that G -templates are generalizations of circulant templates.

Theorem 3.45. Assume that \mathbf{X} is an $m \times n$ array and that $R = V$, the von Neumann configuration. The pair (\mathbf{X}, V) simulates $G = \mathbf{Z}_m \times \mathbf{Z}_n$ (written additively). Furthermore, $G_{\mathbf{X}} = C_{\mathbf{X}}$, the set of all circulant templates on \mathbf{X} .

Proof.

Let $\Delta = \{ (1,0), (m-1,0), (0,1), (0,n-1), (0,0) \}$. Note that \mathbf{X} is isomorphic to $\mathbf{Z}_m \times \mathbf{Z}_n$ where the operation on \mathbf{X} is addition mod (m,n) . Hence we can take $\alpha : \mathbf{X} \rightarrow G$ to be the identity mapping and we only need to show that if $(\mathbf{x}, \mathbf{y}) \in E(D(V))$ then $(\mathbf{x}, \mathbf{y}) \in E(D_{\Delta}(G))$. Assume that $(\mathbf{x}, \mathbf{y}) \in E(D(V))$. Then $\mathbf{x} \in V(\mathbf{y})$ so $(\mathbf{y} - \mathbf{x}) \pmod{(m,n)} \in \Delta$ which implies that $\mathbf{y} = (\mathbf{x} + \mathbf{g}) \pmod{(m,n)}$ for some $\mathbf{g} \in \Delta$. This implies that $(\mathbf{x}, \mathbf{y}) \in E(D_{\Delta}(G))$. Thus (\mathbf{X}, V) simulates G .

To see that $G_{\mathbf{X}} = C_{\mathbf{X}}$, recall that the set of circulant translations on \mathbf{X} is a group under composition which is isomorphic to G and therefore to the group of color-preserving automorphisms of $D_{\Delta}(G)$ for any Δ . Thus, a mapping $\phi : \mathbf{X} \rightarrow \mathbf{X}$ is a color-preserving automorphism if and only if it is a circulant translation. By definition, $T \in C_{\mathbf{X}}$ if and only if $t_{\phi(\mathbf{x})}(\phi(\mathbf{y})) = t_{\mathbf{x}}(\mathbf{y})$ holds for every $\mathbf{x}, \mathbf{y} \in \mathbf{X}$. Thus $C_{\mathbf{X}} = G_{\mathbf{X}}$.

Q.E.D.

In a similar fashion, the set of all G-templates can be shown to be isomorphic to the group algebra of G over \mathbb{C} .

Lemma 3.46. *Assume that (\mathbf{X}, R) simulates G. Let $A \in \mathbb{C}^{\mathbf{X}}$ and define $T \in L_{\mathbf{X}}$ by $T(\mathbf{e}) \equiv A$ and $T(\mathbf{x}) \equiv \phi_{\mathbf{x}}^{-1}(A) \equiv \{ (z, a(\phi_{\mathbf{x}}^{-1}(z))) \}$ where $\phi_{\mathbf{x}}$ is the color preserving automorphism corresponding to \mathbf{x} . The template T is a G-template.*

Proof.

Let $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ and let $\phi_{\mathbf{z}}$ be a color preserving automorphism of $D(R)$. Note that

$$\phi_{\phi_{\mathbf{z}}(\mathbf{x})}^{-1}(\phi_{\mathbf{z}}(\mathbf{y})) = \phi_{(\phi_{\mathbf{z}}(\mathbf{x}))^{-1}}(\phi_{\mathbf{z}}(\mathbf{y}))$$

since the mapping $\mathbf{u} \rightarrow \phi_{\mathbf{u}}$ is a group isomorphism. Therefore, by definition,

$$\phi_{\phi_{\mathbf{z}}(\mathbf{x})}^{-1}(\phi_{\mathbf{z}}(\mathbf{y})) = (\phi_{\mathbf{z}}(\mathbf{x}))^{-1}\phi_{\mathbf{z}}(\mathbf{y}) = (\mathbf{zx})^{-1}\mathbf{zy} = \mathbf{x}^{-1}\mathbf{y} = \phi_{\mathbf{x}}^{-1}(\mathbf{y}).$$

Hence

$$t_{\phi_{\mathbf{z}}(\mathbf{x})}(\phi_{\mathbf{z}}(\mathbf{y})) = a(\phi_{\phi_{\mathbf{z}}(\mathbf{x})}^{-1}(\phi_{\mathbf{z}}(\mathbf{y}))) = a(\phi_{\mathbf{x}}^{-1}(\mathbf{y})) = t_{\mathbf{x}}(\mathbf{y})$$

which shows that T is a G-template and therefore proves the lemma.

Q.E.D.

Theorem 3.47. *The set $G_{\mathbf{X}}$ is an algebra over \mathbb{C} .*

Proof.

Let $S, T \in G_{\mathbf{X}}$ and $r \in \mathbb{C}$. Since addition and scalar multiplication are defined pointwise, $S+T$ and $rS \in G_{\mathbf{X}}$. The templates O and E are G-templates. Hence, we only need to show that $G_{\mathbf{X}}$ is closed under \oplus since the other properties required of an algebra are inherited from $L_{\mathbf{X}}$. Let $R = S \oplus T$, $\mathbf{x}, \mathbf{y} \in \mathbf{X}$, and ϕ be a color preserving automorphism of $D(R)$. By definition of \oplus ,

$$r_{\phi(x)}(\phi(y)) = \sum_{z \in T(\phi(x))} t_{\phi(x)}(z) s_z(\phi(y)) .$$

Since S and T are G -templates,

$$\phi^{-1}T(\phi(x)) = T(x)$$

and

$$t_{\phi(x)}(z) = t_x(\phi^{-1}(z)) \quad \text{and} \quad s_z(\phi(y)) = s_{\phi^{-1}(z)}(y) .$$

Hence

$$\begin{aligned} r_{\phi(x)}(\phi(y)) &= \sum_{z \in T(\phi(x))} t_x(\phi^{-1}(z)) s_{\phi^{-1}(z)}(y) = \\ &= \sum_{z \in \phi^{-1}T(\phi(x))} t_x(z) s_z(y) = \\ &= \sum_{z \in T(x)} t_x(z) s_z(y) = r_x(y) . \end{aligned}$$

which shows that $S \oplus T \in G_X$ and proves the theorem.

Q.E.D.

Definition 3.48. The *group algebra* of $G \equiv X$ over C , denoted $C[G]$, is defined to be the set $C[G] \equiv \{ (a(x_0), a(x_1), \dots, a(x_{n-1})) : a(x_i) \in C \}$ equipped with the usual vector space operations of addition and scalar multiplication; $C[G]$ is also equipped with a vector multiplication defined by a convolution which is tied to the group structure. That is, if

$$\vec{a} = (a(x_0), a(x_1), \dots, a(x_{n-1})), \quad \vec{b} = (b(x_0), b(x_1), \dots, b(x_{n-1})) \in C[G],$$

then $\vec{c} = \vec{a} * \vec{b}$ is defined by

$$c(x_i) = \sum_{u \in X} a(u) b(u^{-1}x).$$

Remark 3.49. Note that $C[G]$ is isomorphic to C^X as a vector space but not necessarily as an algebra.

The following example helps to put the group algebra into perspective.

Example 3.50. Let $G = \mathbb{Z}_n$ be the cyclic group of order n written additively. Thus, if $u \in G$, then $u^{-1} = -u$. If $\vec{c} = \vec{a} * \vec{b}$ then

$$c(x) = \sum_{u \in G} a(u)b((x-u) \pmod n).$$

Thus, the group algebra in this case is an algebraic object which models standard cyclic convolution, or equivalently, polynomial multiplication mod $x^n - 1$. If we take $G = \mathbb{Z}$ then we obtain standard linear convolution.

Theorem 3.51. Assume that (X, R) simulates G . G_X is isomorphic, as a ring and as a vector space over \mathbb{C} , to $\mathbb{C}[G]$.

Proof.

We identify X with G . Recall that a G -template is well defined by defining it at one point. Define $\Gamma : G_X \rightarrow \mathbb{C}[G]$ by

$$\Gamma(T) \equiv (a(x_0), a(x_1), \dots, a(x_{n-1})), \text{ where } T(e) = A.$$

The map Γ is onto since, by Lemma 3.46, given A we can always define a G -template such that $T(e) = A$. To see that Γ is 1-1 assume that $\Gamma(T) = \Gamma(S)$. Then $T(e) = S(e)$ which implies that $S = T$. Since addition and scalar multiplication are defined pointwise on both G_X and $\mathbb{C}[G]$, $S, T \in G_X$ and $r \in \mathbb{C}$ implies that $\Gamma(S+T) = \Gamma(S) + \Gamma(T)$ and $r\Gamma(T) = \Gamma(rT)$. Let $A \equiv S(e)$, $B \equiv T(e)$, and $C \equiv (S \oplus T)(e)$. By definition of \oplus ,
$$c(x) = \sum_{y \in T(e)} t_e(y)s_y(x). \quad \text{Denote} \quad \Gamma(S) * \Gamma(T) \equiv (d(x_0), d(x_1), \dots, d(x_{n-1})).$$
 By definition of multiplication in $\mathbb{C}[G]$, $d(x) = \sum_{y \in X} a(y)b(y^{-1}x)$. Since $S \in G_X$, $s_y(x) = s_e(\phi_{y^{-1}}(x)) = s_e(y^{-1}x) = b(y^{-1}x)$. Hence, since $t_e(y) = a(y)$, $c(x) = \sum_{y \in T(e)} a(y)b(y^{-1}x) = \sum_{y \in X} a(y)b(xy^{-1}) = d(x)$ which shows that

$$\Gamma(S) * \Gamma(T) = \Gamma(S \oplus T).$$

Q.E.D.

Remark 3.52. In the case that $G = \mathbb{Z}_m \times \mathbb{Z}_n$, $\Gamma = \Gamma_0$ defined in Section 3.2.

Group algebras arise in the study of linear representations of groups. These representations furnish another description of the set of G -templates. They are also a source of necessary and sufficient conditions for determining the invertibility of G -templates.

Definition 3.53. A *linear representation* of a group G over \mathbb{C} is a group homomorphism $\alpha : G \rightarrow M_k(\mathbb{C})$ for some $k \geq 1$. A *linear representation* of the group algebra $\mathbb{C}[G]$ over \mathbb{C} is an algebra homomorphism $\alpha : \mathbb{C}[G] \rightarrow M_k(\mathbb{C})$; that is, α preserves sums, products, and scalar multiplications.

Denote $G = \{x_0, x_1, \dots, x_{n-1}\}$. Note that for each $k \in [0, n-1]$ we can define a permutation $\tau_k \in \Sigma_n$ by the rule $\tau_k(i) = j$ if and only if $x_i x_k = x_j$. Henceforth, we use the symbols τ_k to denote these particular permutations.

Definition 3.54. A *right regular representation* of G is a representation $\alpha : G \rightarrow M_k(\mathbb{C})$ of the form $\alpha(x_k) \equiv P_{\tau_k}$. A *right regular representation* of $\mathbb{C}[G]$ is a representation $\alpha^* : \mathbb{C}[G] \rightarrow M_k(\mathbb{C})$ of the form

$$\alpha^*((a(x_0), a(x_1), \dots, a(x_{n-1}))) \equiv \sum_{i=0}^{n-1} a(x_i) \alpha(x_i) = \sum_{i=0}^{n-1} a(x_i) P_{\tau_k}$$

where α is a right regular representation of G .

It is a fact that right regular representations are isomorphisms. Note that there is a different right regular representation of G for every different ordering of G . Left regular

representations defined relative to different orderings are equivalent up to permutations [19].

Remark 3.55. Any representation α of G can be extended to a representation α^* of $\mathbb{C}[G]$ in a fashion similar to that in Definition 3.54. In fact, G can be considered to be embedded in $\mathbb{C}[G]$ via the mapping $\mathbf{x}_i \rightarrow \vec{e}_{i+1}$ where \vec{e}_{i+1} represents the i^{th} element of the standard basis. Hence, we can define $\alpha^*(\vec{e}_i) \equiv \alpha(\mathbf{x}_i)$ and then extend by linearity. It can be shown that the multiplication is preserved [27]. Henceforth, we use the same name for either representation, that is, we take $\alpha^* \equiv \alpha$.

Theorem 3.56. Let $\mathbf{X} = \{ \mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1} \}$ be any ordering of \mathbf{X} and let $\Psi : L_{\mathbf{X}} \rightarrow M_n(\mathbb{C})$ be defined relative to this ordering. Let $\vec{a} \in \mathbb{C}[G]$ and let $T_{\vec{a}}$ denote the G -template associated with \vec{a} , that is, $T_{\vec{a}} = \Gamma^{-1}(\vec{a})$, and $M_{\vec{a}}$ the value of the right regular representation at \vec{a} , that is, $M_{\vec{a}} = \alpha(\vec{a})$, where α is the right regular representation of $\mathbb{C}[G]$. Then $\Psi(T_{\vec{a}}) = M_{\vec{a}}$, that is, $\Psi = \alpha\Gamma$.

Proof.

Let $A \equiv T_{\vec{a}}(\mathbf{e})$. Then $t_{\mathbf{x}_i}(\mathbf{x}_j) = t_{\mathbf{e}}(\phi_{\mathbf{x}_i}^{-1}(\mathbf{x}_j)) = a(\mathbf{x}_i^{-1}\mathbf{x}_j)$. Hence, $\Psi(T_{\vec{a}}) = (\mu_{ij})$ where $\mu_{ij} = a(\mathbf{x}_i^{-1}\mathbf{x}_j)$. Denote $M_{\vec{a}} \equiv (\gamma_{ij})$ and $P_{\tau_k} = (p_{ij}^{(k)})$ for $k \in [0, n-1]$. Assume that $p_{ij}^{(k)} = 1$ for some triple (i, j, k) . Then $j = \tau_k(i)$ or $\mathbf{x}_i\mathbf{x}_k = \mathbf{x}_j$. Furthermore, if $p_{ij}^{(h)} = 1$, then $\mathbf{x}_i\mathbf{x}_h = \mathbf{x}_j$ so $\mathbf{x}_h = \mathbf{x}_i^{-1}\mathbf{x}_j = \mathbf{x}_k$ which implies that $h = k$. Thus, by definition of the right regular representation of $\mathbb{C}[G]$, each γ_{ij} is of the form $a(\mathbf{x}_k)$ for some k . In fact, $\gamma_{ij} = a(\mathbf{x}_k)$ if and only if $p_{ij}^{(k)} = 1$ which in turn is true if and only if $\mathbf{x}_i\mathbf{x}_k = \mathbf{x}_j$. Thus, $\gamma_{ij} = a(\mathbf{x}_i^{-1}\mathbf{x}_j) = t_{\mathbf{x}_i}(\mathbf{x}_j) = \mu_{ij}$ which implies that $M_{\vec{a}} = \Psi(T_{\vec{a}})$ which is the desired result.

Q.E.D.

An alternative statement of Theorem 3.56 is that the diagram in Figure 3.3 commutes if Ψ and α are computed relative to the same ordering of G .

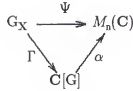


Figure 3.3. Commutative diagram representing Theorem 3.56.

Definition 3.57. We say that $T \in L_X$ is an *elementary template* if and only if for every $x, y \in X$, $t_x(y) \in \{0, 1\}$.

Lemma 3.58. Assume that $T \in G_X$ is an elementary template. The sets $\{T(x) : x \in X\}$ are all translates of some set $S \subset X$, that is, there exists $S \subset X$ such that for every $x, y \in X$, $xS = T(x)$ where $xS = \{xs : s \in S\}$. Furthermore, for every $S \subset X$, there exists a unique elementary template $T \in G_X$ such that for every $x, y \in X$, $xS = T(x)$. We say that T (or S) is induced by S (or T).

Proof.

Define $S \equiv T(e)$ and let $x \in X$. Then

$$\begin{aligned}
 xS &= \{y \in X : y = xs \text{ for some } s \in S\} = \\
 &= \{y \in X : x^{-1}y \in T(e)\} = \\
 &= \{y \in X : t_e(x^{-1}y) \neq 0\} = \\
 &= \{y \in X : t_x(y) \neq 0\} = T(x).
 \end{aligned}$$

The converse follows immediately from Lemma 3.46

Q.E.D.

Remark 3.59. The image $T(e)$ is the graph of the characteristic function of S . Hence, by Theorem 3.56, $\Psi(T) = \sum_{s \in S} \alpha(s)$ where α denotes the right regular representation of G .

Remark 3.60. Such a set S exists for any G -template.

We now give the definition of the discrete Radon transform.

Definition 3.61. Let G be a finite group and let $S \subset G$. For every $f : G \rightarrow \mathbb{C}$ define $\bar{f} : G \rightarrow \mathbb{C}$ by $\bar{f}(t) \equiv \sum_{s \in tS} f(s)$. The transform $f \rightarrow \bar{f}$ is called the *discrete Radon transform* based on S .

Theorem 3.62. Assume that (X, R) simulates G . Let $S \subset G$ and let $T \in G_X$ be induced by S . The discrete Radon transform based on S is invertible if and only if T is.

Proof.

We identify X with G . Let $f : X \rightarrow \mathbb{C}$ and let F be the graph of f . By definition of the discrete Radon transform, $\bar{f}(x) = \sum_{s \in xS} f(s) = \sum_{s \in T(x)} f(s)$. Hence, if we denote by \bar{F} the graph of \bar{f} , then we have that $\bar{F} = F \oplus T$. Therefore the discrete Radon transform based on S is invertible if and only if the mapping $F \rightarrow F \oplus T$ is invertible. This will be the case if and only if T is invertible.

Q.E.D.

The following definitions are expressed in terms of group representations. They can just as easily be made for representations of group algebras and we will consider this to be done.

Definition 3.63. Let $\alpha : G \rightarrow M_k(\mathbb{C})$ be a representation of G . We say that α is *reducible* if and only if there exists a nonsingular matrix $M \in M_k(\mathbb{C})$ such that for every $x \in G$

$$\alpha(x) = M \begin{bmatrix} A(x) & B(x) \\ 0 & C(x) \end{bmatrix} M^{-1}.$$

If α is not reducible, then we say that α is *irreducible*.

Definition 3.64. We say that two representations α and β of G are *similar* if and only if there exists an invertible matrix $M \in M_k(\mathbb{C})$ such that for every $x \in G$, $\alpha(x) = M\beta(x)M^{-1}$.

Fact 3.65. If $\alpha : G \rightarrow M_n(\mathbb{C})$ is a right regular representation of G , then there exists a nonsingular matrix M and irreducible representations $\alpha_1, \alpha_2, \dots, \alpha_k$ of G such that for every $g \in G$

$$\alpha(g) = M \begin{bmatrix} \alpha_1(g) & & & \\ & \alpha_2(g) & 0 & \\ & & \ddots & \\ & 0 & & \ddots & \\ & & & & \alpha_k(g) \end{bmatrix} M^{-1}.$$

Moreover, if ρ is any irreducible representation then ρ is similar to α_i for some i [27, 29].

The same statement holds true for $\mathbb{C}[G]$

Theorem 3.66. Let $S \subset G$ and let $T \in G_X$ be induced by S . The discrete Radon transform based on S is invertible if and only if $\rho(\Gamma(T))$ is invertible for every irreducible representation ρ of $\mathbb{C}[G]$.

Proof.

Let $\Gamma(T) = \bar{p}$. Assume that the discrete Radon transform based on S is invertible and let ρ be any irreducible representation of $\mathbb{C}[G]$. By Theorem 3.62, T is invertible.

Let α be a right regular representation of G and let $\Psi : L_X \rightarrow M_n(\mathbf{C})$ be defined relative to the same ordering of G as α . By Theorem 3.56, $\alpha(\bar{p}) = \Psi(T)$ which implies that $\alpha(\bar{p})$ is invertible. By Fact 3.65, there exists a nonsingular matrix $M \in M_n(\mathbf{C})$, and irreducible representations $\alpha_1, \alpha_2, \dots, \alpha_k$ of $\mathbf{C}[G]$ such that

$$\Psi(T) = M \begin{bmatrix} \alpha_1(\bar{p}) & & & \\ & \alpha_2(\bar{p}) & 0 & \\ & 0 & \ddots & \\ & & & \alpha_k(\bar{p}) \end{bmatrix} M^{-1}.$$

Furthermore, there exists an $i \in [1, k]$ such that $\rho = \alpha_i$. Since $\Psi(T)$ is invertible if and only if every $\alpha_i(\bar{p})$ is, it follows that $\rho(\bar{p}) = \rho(\Gamma(T))$ is invertible.

Conversely, assume that $\rho(\Gamma(T))$ is invertible for every irreducible representation ρ of $\mathbf{C}[G]$. Then, by Fact 3.65, $\Psi(T)$ is invertible. Hence, T is invertible so by Theorem 3.62, the discrete Radon transform based on S is invertible.

Q.E.D.

Note that the proof of Theorem 3.66 yields the more general result given in the next corollary.

Corollary 3.67. *Let $T \in G_X$. T is invertible if and only if $\rho(\Gamma(T))$ is invertible for every irreducible representation ρ of $\mathbf{C}[G]$.*

Definition 3.68. If ρ is a representation of G , then the *contragredient representation* of ρ is the representation defined by $\hat{\rho}(\mathbf{x}) \equiv \rho(\mathbf{x}^{-1})^t$. Note that $\rho(\mathbf{x}) = \hat{\rho}(\mathbf{x}^{-1})^t$ and that every representation is the contragredient representation of some representation.

Lemma 3.69. *Let $S \subset G$, let $T \in G_X$ be induced by S , and let ρ be any irreducible*

representation of $\mathbf{C}[G]$. The matrix $\rho(\Gamma(T))$ is invertible if and only if $\sum_{\mathbf{s} \in S} \hat{\rho}(\mathbf{s}^{-1})$ is.

Proof.

Let $\vec{a} = \Gamma(T)$. By Remark 3.56, we can write $\rho(\vec{a}) = \sum_{i=0}^{n-1} a(\mathbf{x}_i) \rho(\mathbf{x}_i)$. Furthermore, since $a : \mathbf{X} \rightarrow \mathbf{C}$ is the characteristic function of S , $\rho(\vec{a}) = \sum_{\mathbf{s} \in S} \rho(\mathbf{s})$. Thus, $\rho(\Gamma(T)) = \rho(\vec{a})$ is invertible if and only if $\sum_{\mathbf{s} \in S} \rho(\mathbf{s})$ is invertible. Moreover $\sum_{\mathbf{s} \in S} \rho(\mathbf{s})$ is invertible if and only if $\sum_{\mathbf{s} \in S} \hat{\rho}(\mathbf{s}^{-1})^t = (\sum_{\mathbf{s} \in S} \hat{\rho}(\mathbf{s}^{-1}))^t$ is. Since a matrix is invertible if and only if its transpose is invertible, we may conclude that the lemma is true.

Q.E.D.

Corollary 3.70. Let $S \subset G$. The discrete Radon transform based on S is invertible if and only if for every irreducible representation ρ of G , the matrix $\sum_{\mathbf{s} \in S} \rho(\mathbf{s}^{-1})$ is invertible.

Proof.

By Theorem 3.66, the discrete Radon transform based on S is invertible if and only if for every irreducible representation $\hat{\rho}$ of G the matrix $\hat{\rho}(\Gamma(T))$ is invertible. By Lemma 3.69, this will happen if and only if $\sum_{\mathbf{s} \in S} \rho(\mathbf{s}^{-1})$ is invertible.

Q.E.D.

Corollary 3.70 was first proved by Diaconis and Graham using different methods [11].

Thus, the invertibility of the discrete Radon transform is linked to the invertibility of G -templates. In the case of circulant templates the criteria is particularly useful since

the irreducible representations of abelian groups are all one-dimensional. In fact, the irreducible representations of $Z_m \times Z_n$ over C are the mn elements of the set $\{\omega_m^i \omega_n^j : i, j \in Z\}$. Hence, if ρ is an irreducible representation of $C[G]$, then $\rho(\Gamma(T)) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} t_0(i, j) \omega_m^i \omega_n^j = p_t(\omega_m^i, \omega_n^j, 0)$. Recall from Section 3.2 that the block circulant matrices with circulant blocks are all simultaneously diagonalizable by the Fourier matrices. This is a special case of Fact 3.65. By the aforementioned observations and Theorem 3.56, the numbers $p_t(\omega_m^i, \omega_n^j, 0)$ must be the eigenvalues of the matrix $\Psi(T)$, which is block circulant with circulant blocks.

It is interesting to point out that using the above criteria for circulant templates can yield conditions for the invertibility of some particular templates very easily whereas for others it is not so easy. For example, if we let M denote the *Moore* configuration, which is just a 3×3 square at every point, then one can use a few trigonometric manipulations to show that M is invertible on the $n \times n$ array X if and only if 3 does not divide n . If one considers the von Neumann configuration, however, it does not appear to be as easy. A computer program has been written which suggests that the discrete Radon transform based on the von Neumann configuration on X is invertible if and only if 5 divides n . We have not been able to prove the necessity of this conjecture.

As we mentioned previously, G -templates have potential applications to parallel processing since they are translation invariant with respect to Cayley color graphs which are being studied as possible models for parallel computer architectures. As mentioned in the Introduction, many parallel computer architectures are built to implement translation invariant transformations since it is generally easier to construct and control such devices. It is reasonable to expect that G -templates will furnish a good algebraic description of computing environments based on Cayley networks.

The topic of group representations has been in existence since about 1896 when Frobenius first defined group characters [29]. It is a well studied area and has seen applications to physics and chemistry as well as in group theory. It is fascinating to think that a subject developed in the early part of the century could have applications to parallel computer architectures. It is possible that this will be another case where the mathematics precedes the application by a significant time period.

We have defined G -templates and shown that they are generalizations of circulant templates. We have also shown that the set of all G -templates is an algebra which is isomorphic to the group algebra of G over \mathbb{C} . We then described the relationship between G -templates and the discrete Radon transform.

PART II

LOCAL DECOMPOSITIONS OF FOURIER TEMPLATES

In Part II we focus on deriving local decompositions of the Fourier template with respect to the von Neumann configuration. We recall that the von Neumann configuration is the template configuration function $V: \mathbf{X} \rightarrow 2^{\mathbf{X}}$ defined on the $m \times n$ array \mathbf{X} by $V(x,y) = \{(x,y), (x+1,y)(\bmod (m,n)), (x-1,y)(\bmod (m,n)), (x,y+1)(\bmod (m,n)), (x,y-1)(\bmod (m,n))\}$. V models a mesh-connected, or rectangular, array of processors with nearest neighbor connections. Henceforth, when we refer to local we shall mean with respect to V . We shall also take \mathbf{X} to be an $m \times n$ array where m and n are positive integers but are not restricted in any other way.

We use matrix algebra associated with the fast Fourier transform (FFT) and numerical linear algebra to derive local decompositions of the Fourier matrices. Due to the existence of the isomorphism $\Psi: L_{\mathbf{X}} \rightarrow M_{mn}(\mathbb{C})$, these decompositions can be interpreted as decompositions of the corresponding templates.

We say that a template $T \in L_{\mathbf{X}}$ is *separable* if $M_T = M_1 \otimes M_2 = (M_1 \otimes I_n)(I_m \otimes M_2)$ where M_1 and M_2 are $m \times m$ and $n \times n$ matrices respectively. We have already observed that the Fourier template has this property. This fact implies that it is sufficient (but not necessary) to seek decompositions which are local with respect to restrictions of the von Neumann configuration to linear arrays. If T is separable, then $T = T_2 \oplus T_1$ where $T_2 = \Psi^{-1}(I_m \otimes M_2)$ and $T_1 = \Psi^{-1}(M_1 \otimes I_n)$. If $A \in \mathbb{C}^{\mathbf{X}}$, then the mapping $A \rightarrow A \oplus T_2$ represents an identical computation being done on each row and $(A \oplus T_2) \oplus T_1$

represents an identical computation being done along each column of the result. Thus, it is sufficient to construct decompositions of T_1 and T_2 with respect to the configurations R_1 and R_2 defined by

$$R_1(x,y) = \{ (x,y), (x+1,y)(\text{mod } (m,n)), (x-1,y)(\text{mod } (m,n)) \}$$

and

$$R_2(x,y) = \{ (x,y), (x,y+1)(\text{mod } (m,n)), (x,y-1)(\text{mod } (m,n)) \}.$$

This can be achieved by factoring the matrices M_1 and M_2 into products of tridiagonal matrices. For this reason, we concern ourselves with developing decompositions of the one-dimensional Fourier matrices into products of tridiagonal matrices and permutation matrices. We call such decompositions *tridiagonal decompositions*. We will describe an algorithm for computing the permutations locally. Henceforth, we shall refer to the one-dimensional Fourier matrices simply as Fourier matrices.

Part II is divided into two chapters. In the first chapter, chapter 4, we use matrix identities associated with FFTs to develop tridiagonal decompositions of the Fourier matrices. In the second chapter, chapter 5, we develop a numerical method for computing tridiagonal decompositions of the Fourier matrices.

CHAPTER 4

TRIDIAGONAL DECOMPOSITIONS OF FOURIER MATRICES BASED ON MATRIX ALGEBRA ASSOCIATED WITH THE FFT

In this chapter, we use matrix identities associated with FFTs to develop tridiagonal decompositions of Fourier matrices. We describe methods for decreasing the number of parallel operations and mapping “large” DFTs onto smaller arrays using two-dimensional techniques. We investigate the parallel time complexity of the algorithms implied by the decompositions.

In the first section, we use identities developed by Rose [48] to derive decompositions of F_n into products of permutation matrices and block diagonal matrices in two different ways. The diagonal blocks of these matrices are of the form F_p where p is a prime divisor of n . We then derive a matrix identity associated with the Rader prime algorithm and the circular convolution theorem [6, 10, 31]. This identity is used to factor the Fourier matrices of prime order into products of permutation matrices, Fourier matrices of lower, composite order, and certain sparse triangular matrices. We show how the sparse matrices that appear can be factored. Taken together, these theorems provide an algorithm for deriving tridiagonal decompositions of F_n for arbitrary n .

In the second section, we derive expressions for the number of parallel arithmetic steps and upper bounds on the number of parallel permutation, or data manipulation, steps required to implement the algorithms resulting from the decompositions. We provide a table giving upper bounds on the number of steps required to implement these algorithms on a mesh-connected array for certain image dimensions. Since many permutation matrices appear in the decompositions, data manipulation steps represent a

significant part of any algorithm derived from these decompositions. Data manipulation steps are important when trying to implement algorithms on massively parallel architectures since the processors do not have a global shared memory. Therefore, particular attention is paid to this aspect of the parallel time complexity.

4.1. Derivation of FFT-Based Decompositions

We first establish notation and state identities which will be required in deriving the tridiagonal decompositions based on FFT identities. Unless otherwise stated we consider matrices as ordered from 0 to $n-1$. We denote by Σ_n the group of permutations of n objects, and by \mathbf{Z}_n the ring of integers modulo n . We think of Σ_n as acting on \mathbf{Z}_n . Let $m \in \mathbf{Z}$ with $m > 1$ and let $\omega_m \equiv \exp(-2\pi i/m)$ where $i = (-1)^{1/2}$. Assume for the remainder of the section that $n = mk$ where $m, k > 1$.

Definition 4.1. Define $\sigma_{mk} : \mathbf{Z}_n \rightarrow \mathbf{Z}_n$ by the following rule:

If $i \in \mathbf{Z}_n$ and $i = a + bm$ with $0 \leq a < m$ and $0 \leq b < k$, then $\sigma_{mk}(i) = ak + b$.

σ_{mk} is called a *shuffle* permutation.

Definition 4.2. Define $P(m, k) \in M_n$ by $P(m, k) \equiv P_{\sigma_{mk}}$.

$P(m, k)$ is called a *shuffle permutation matrix*. Note that $P(m, k) = P(k, m)^t = P(k, m)^{-1}$.

Definition 4.3. Define the $n \times n$ diagonal matrix $D(m, k)$ by

$$D(m, k) \equiv \text{blockdiag}[D_j(m, k)], \quad j=0, 1, \dots, k-1,$$

where

$$D_j(m, k) \equiv \text{diag}(\omega_n^{ij}), \quad i=0, 1, \dots, m-1, \quad j=0, 1, \dots, k-1.$$

These matrices are called *twiddle matrices* or *twiddle factors*.

For any $p, k > 1$ we denote $D_{p,k} \equiv D(p^{k-1}, p)$ and $P_{p,k} \equiv P(p^{k-1}, p)$. For convenience, we take $P(n, 1) \equiv I_n$, the $n \times n$ identity matrix, and $I_1 \equiv 1$.

It is useful to have other descriptions of the twiddle matrices. We give two more here; the first is useful for expressing matrix identities whereas the second is better for implementation purposes.

Definition 4.4. Let A be any $m \times m$ matrix and define $T_k(A)$ by

$$T_k(A) \equiv \text{blockdiag}[A^0, A^1, A^2, \dots, A^{k-1}].$$

Let

$$\Omega_m \equiv \text{diag}(1, \omega_n, \omega_n^2, \dots, \omega_n^{m-1})$$

and define

$$\hat{D}(m, k) \equiv T_k(\Omega_m).$$

Lemma 4.5. $D(m, k) = \hat{D}(m, k)$.

Proof.

The lemma follows immediately from the fact that $D_j(m, k) = \Omega_m^j$.

Q.E.D.

Lemma 4.6. If $D(m, k) = \text{diag}(d_i)$, then $d_i = \omega_n^{r_i q}$, where r and q are the unique integers satisfying $i = qm + r$ with $0 \leq r < m$, that is, $r = i \pmod{m}$ and $q = \frac{(i-r)}{m}$.

Proof.

By Lemma 4.5, d_i is the (r, r) entry of the diagonal matrix $\Omega_m^q = \text{diag}(1, \omega_n^q, \omega_n^{2q}, \dots, \omega_n^{(m-1)q})$. Thus, $d_i = \omega_n^{r_i q}$.

We now state some basic identities concerning Fourier matrices, Kronecker products, and permutation matrices which were developed by Rose [48]. We will use these identities as building blocks for some of our derivations.

Let C_n denote the $n \times n$ circulant permutation matrix

$$C_n \equiv \text{circ}(0,0,\dots,0,1).$$

For any integer s , let $Q(m,k,s)$ denote the $n \times n$ permutation matrix

$$Q(m,k,s) \equiv P(k,m)T_k(C_m^s)P(m,k).$$

Facts.

1.) If A is an $n \times n$ matrix and B is an $m \times m$ matrix, then $P(n,m)(A \otimes B)P(m,n) = B \otimes A$.

2.) (General Radix Identity)

$$F_n = (F_m \otimes I_k)D(k,m)(I_m \otimes F_k)P(k,m)$$

3.) (Twiddle Free Identity) Assume that m and k are relatively prime and let $m^* \equiv m^{-1}(\text{mod } k)$ and $k^* \equiv k^{-1}(\text{mod } m)$. Then

$$F_n = Q(m,k,-k^*)(F_m \otimes F_k)T_m(C_k^{m^*})P(k,m).$$

We now use these identities to derive decompositions of the Fourier matrices into products of permutation matrices and block diagonal matrices, the blocks of which are Fourier matrices of prime order. We first assume that n is not a power of a prime.

Theorem 4.7. If $n = \prod_{i=1}^s p_i^{k_i}$ with $s \geq 2$ and

$$n_j = \begin{cases} \prod_{i=1}^j p_i^{k_i} & \text{if } 1 \leq j \leq s \\ 1 & \text{if } j = -1, 0 \end{cases},$$

$c_j = n/n_j$ for $-1 \leq j \leq s$, $q_j = p_j^{k_j}$ for $1 \leq j \leq s$, and $q_0 = 1$, then there exists permutation matrices Q_1, Q_2, \dots, Q_{s-2} , and H such that

$$F_n = \left[\prod_{j=0}^{s-2} (I_{n_j} \otimes Q_{2j+1}) (I_{n_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}}) \right] (I_{n_{s-1}} \otimes F_{q_s}) \left[\prod_{j=1}^{s-1} (I_{n_{s-j-1}} \otimes Q_{2(s-j)}) \right].$$

Proof.

The proof is by induction on the number of primes s .

Assume $s = 2$. Then, since $c_1 = q_2$, $n_1 = q_1$, and $n_0 = 1$, by Fact 3.), there exists permutation matrices Q_1 and Q_2 , namely $Q_1 = Q(q_1, q_2, -q_2^*)$ and $Q_2 = T_{q_1}(C_{q_2}^{q_1^*})P(q_2, q_1)$, such that $F_n = Q_1(F_{q_1} \otimes I_{q_2})(I_{q_1} \otimes F_{q_2})Q_2$ which is exactly the statement of the theorem.

Assume that the theorem is true for all integers with prime factorization of length less than s . In particular, the theorem is true for c_1 . Thus, letting $\hat{n}_j = \prod_{i=2}^j p_i^{k_i}$ and $\hat{n}_1 = 1$, we have that there exists $Q_3, \dots, Q_{2s-3}, Q_4, \dots, Q_{2s-2}$ such that

$$F_{c_1} = \left[\prod_{j=1}^{s-2} (I_{\hat{n}_j} \otimes Q_{2j+1}) (I_{\hat{n}_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}}) \right] (I_{\hat{n}_{s-1}} \otimes F_{q_s}) \left[\prod_{j=2}^{s-2} (I_{\hat{n}_{s-j-1}} \otimes Q_{2(s-j)}) \right].$$

Moreover, there exists Q_1 and Q_2 such that

$$F_n = Q_1(F_{q_1} \otimes I_{c_1})(I_{q_1} \otimes F_{c_1})Q_2.$$

Putting these equations together and observing that $\hat{n}_j q_1 = n_j$ yields

$$F_n = Q_1(F_{q_1} \otimes I_{c_1})$$

$$\begin{aligned}
& I_{q_1} \otimes \left[\left[\prod_{j=1}^{s-2} (I_{\hat{n}_j} \otimes Q_{2j+1})(I_{\hat{n}_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}}) \right] (I_{\hat{n}_{s-1}} \otimes F_{q_s}) \left[\prod_{j=2}^{s-2} (I_{\hat{n}_{s-j-1}} \otimes Q_{2(s-j)}) \right] \right] Q_2 = \\
& = (I_{n_0} \otimes Q_1)(I_{n_0} \otimes F_{q_1} \otimes I_{c_1}) \\
& \left[\prod_{j=1}^{s-2} (I_{n_j} \otimes Q_{2j+1})(I_{n_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}}) \right] (I_{n_{s-1}} \otimes F_{q_s}) \left[\prod_{j=2}^{s-2} (I_{n_{s-j-1}} \otimes Q_{2(s-j)}) \right] (I_{n_0} \otimes Q_2) = \\
& = \left[\prod_{j=0}^{s-2} (I_{n_j} \otimes Q_{2j+1})(I_{n_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}}) \right] (I_{n_{s-1}} \otimes F_{q_s}) \left[\prod_{j=1}^{s-1} (I_{n_{s-j-1}} \otimes Q_{2(s-j)}) \right]
\end{aligned}$$

which is the desired result.

Q.E.D.

Corollary 4.8. *Using the notation of Theorem 4.7 we have that*

$$F_n = \left[\prod_{j=0}^{s-2} R_j E_j \right] (I_{n_{s-2}} \otimes P(q_{s-1}, c_{s-1}))(I_{n_{s-1}} \otimes F_{q_s}) \left[\prod_{j=1}^{s-1} (I_{n_{s-j-1}} \otimes Q_{2(s-j)}) \right]$$

where

$$R_j = (I_{n_{j-1}} \otimes P(q_j, c_j))(I_{n_j} \otimes Q_{2j+1})(I_{n_j} \otimes P(c_{j+1}, q_{j+1}))$$

and

$$E_j = (I_{n_j c_{j+1}} \otimes F_{q_{j+1}}).$$

Proof.

Use the fact that

$$I_{n_j} \otimes F_{q_{j+1}} \otimes I_{c_{j+1}} = (I_{n_j} \otimes P(c_{j+1}, q_{j+1}))(I_{n_j c_{j+1}} \otimes F_{q_{j+1}})(I_{n_j} \otimes P(q_{j+1}, c_{j+1}))$$

in Theorem 4.7.

Q.E.D.

One can also use the general radix identity in a similar way to obtain alternative decompositions.

Theorem 4.9. Assume that $n = \prod_{j=1}^s k_j$ is any nontrivial factorization of n as a product of

positive integers. Let $n_i \equiv \prod_{j=1}^i k_j$, $n_0 \equiv 1$, and $c_i \equiv \frac{n}{n_i}$. Then

$$F_n = \left[\prod_{j=1}^{s-1} (I_{n_{j-1}} \otimes P(c_j, k_j)) (I_{n_{j-1}c_j} \otimes F_{k_j}) (I_{n_{j-1}} \otimes P(k_j, c_j)) (I_{n_{j-1}} \otimes D(c_j, k_j)) \right] \\ \left[I_{n_{s-1}} \otimes F_{k_s} \right] \left[\prod_{j=2}^s (I_{n_{j-1}} \otimes P(c_{s-j+1}, k_{s-j+1})) \right].$$

Proof.

The proof is by induction on s , the number of factors. If $s = 2$, then $n_1 = k_1$ and $c_1 = k_2$. Therefore, the statement of the theorem is that

$$F_n = (I_{n_0} \otimes P(c_1, k_1)) (I_{n_0c_1} \otimes F_{k_1}) (I_{n_0} \otimes P(k_1, c_1)) (I_{n_0} \otimes D(c_1, k_1)) (I_{n_1} \otimes F_{k_2}) (I_{n_0} \otimes P(c_1, k_1)) = \\ = P(k_2, k_1) (I_{k_2} \otimes F_{k_1}) P(k_1, k_2) D(k_2, k_1) (I_{k_1} \otimes F_{k_2}) P(k_2, k_1) = \\ = (F_{k_1} \otimes I_{k_2}) D(k_2, k_1) (I_{k_1} \otimes F_{k_2}) P(k_2, k_1)$$

which is precisely the general radix identity.

Assume that the theorem is true for $2, 3, \dots, s-1$ for some $s > 2$ and that n is as in the statement of the theorem for this s . With this assumption, the theorem is true for

$c_1 = \prod_{j=2}^s k_j$. Therefore, letting $\hat{n}_j \equiv \frac{n_{j+1}}{k_1}$, $n_0 \equiv 1$, and $\hat{c}_j \equiv \frac{c_1}{\hat{n}_j}$ for $j \in [1, s-1]$ and using

the facts that $\hat{n}_j n_1 = n_{j+1}$ and $\hat{c}_j = c_{j+1}$ we have that

$$F_{c_1} = \left[\prod_{j=1}^{s-2} (I_{\hat{n}_{j-1}} \otimes P(\hat{c}_j, k_{j+1})) (I_{\hat{n}_{j-1}\hat{c}_j} \otimes F_{k_{j+1}}) (I_{\hat{n}_{j-1}} \otimes P(k_{j+1}, \hat{c}_j)) (I_{\hat{n}_{j-1}} \otimes D(\hat{c}_j, k_{j+1})) \right] \\ \left[I_{\hat{n}_{s-2}} \otimes F_{k_{s-1}} \right] \left[\prod_{j=2}^{s-1} (I_{\hat{n}_{s-j-1}} \otimes P(\hat{c}_{s-j}, k_{s-j-1})) \right].$$

Hence

$$I_{k_1} \otimes F_{c_1} = I_{n_1} \otimes F_{c_1} = \\ = \left[\prod_{j=1}^{s-2} (I_{\hat{n}_{j-1}n_1} \otimes P(c_{j+1}, k_{j+1})) (I_{\hat{n}_{j-1}n_1c_{j+1}} \otimes F_{k_{j+1}}) (I_{\hat{n}_{j-1}n_1} \otimes P(k_{j+1}, c_{j+1})) (I_{\hat{n}_{j-1}n_1} \otimes D(c_{j+1}, k_{j+1})) \right] \\ = \left[I_{\hat{n}_{s-2}n_1} \otimes F_{k_s} \right] \left[\prod_{j=2}^{s-1} (I_{n_1\hat{n}_{s-j-1}} \otimes P(c_{s-j-1}, k_{s-j-1})) \right] = \\ \left[\prod_{j=2}^{s-1} (I_{n_{j-1}} \otimes P(c_j, k_j)) (I_{n_{j-1}c_j} \otimes F_{k_j}) (I_{n_{j-1}} \otimes P(k_j, c_j)) (I_{n_{j-1}} \otimes D(c_j, k_j)) \right] \\ \left[I_{n_{s-1}} \otimes F_{k_s} \right] \left[\prod_{j=2}^{s-1} (I_{n_{s-j}} \otimes P(c_{s-j-1}, k_{s-j-1})) \right].$$

Therefore

$$F_n = (F_{k_1} \otimes I_{c_1}) D(c_1, k_1) (I_{k_1} \otimes F_{c_1}) P(c_1, k_1) \\ = P(c_1, k_1) (I_{c_1} \otimes F_{k_1}) P(k_1, c_1) D(c_1, k_1) (I_{k_1} \otimes F_{c_1}) P(c_1, k_1) = \\ = (I_{n_0} \otimes P(c_1, k_1)) (I_{n_0c_1} \otimes F_{k_1}) (I_{n_0} \otimes P(k_1, c_1)) (I_{n_0} \otimes D(c_1, k_1)) \\ \left[\prod_{j=2}^{s-1} (I_{n_{j-1}} \otimes P(c_j, k_j)) (I_{n_{j-1}c_j} \otimes F_{k_j}) (I_{n_{j-1}} \otimes P(k_j, c_j)) (I_{n_{j-1}} \otimes D(c_j, k_j)) \right]$$

$$\begin{aligned}
& \left[I_{n_{s-1}} \otimes F_{k_s} \right] \left[\prod_{j=2}^{s-1} (I_{n_{s-j}} \otimes P(c_{s-j-1}, k_{s-j-1})) \right] \left[(I_{n_0} \otimes P(c_1, k_1)) \right] = \\
& = \left[\prod_{j=1}^{s-1} (I_{n_{j-1}} \otimes P(c_j, k_j)) (I_{n_{j-1}c_j} \otimes F_{k_j}) (I_{n_{j-1}} \otimes P(k_j, c_j)) (I_{n_{j-1}} \otimes D(c_j, k_j)) \right] \\
& \left[I_{n_{s-1}} \otimes F_{k_s} \right] \left[\prod_{j=2}^s (I_{n_{s-j}} \otimes P(c_{s-j+1}, k_{s-j+1})) \right].
\end{aligned}$$

which is the desired result.

Q.E.D.

Corollary 4.10. Let n_i, n_i , and c_i be as in Theorem 4.7 and denote

$$P_i = P(c_{i+1}, p_{i+1}^{k_{i+1}}), \quad \text{and} \quad D_i = D(c_{i+1}, p_{i+1}^{k_{i+1}}).$$

Then

$$\begin{aligned}
F_n &= \left[\prod_{j=1}^{s-1} (I_{n_{j-1}} \otimes P_{j-1})(I_{n_{j-1}c_j} \otimes F_{p_j^{k_j}})(I_{n_{j-1}} \otimes P_{j-1}^t)(I_{n_{j-1}} \otimes D_{j-1}) \right] \\
&\left[I_{n_{s-1}} \otimes F_{p_s^{k_s}} \right] \left[\prod_{j=2}^s (I_{n_{s-j}} \otimes P_{s-j}) \right].
\end{aligned}$$

Corollary 4.11. If $k \geq 2$, then $F_{p^k} = G_p(I_{p^{k-1}} \otimes F_p)H_p$ where

$$G_p = \prod_{m=0}^{k-2} \left[(I_{p^m} \otimes P_{p^{k-m}})(I_{p^{k-1}} \otimes F_p)(I_{p^m} \otimes P_{p^{k-m}}^t)(I_{p^m} \otimes D_{p^{k-m}}) \right]$$

and

$$H_p = \prod_{m=2}^k \left[I_{p^{k-m}} \otimes P_{p^m} \right].$$

Note that Corollary 4.11 results in tridiagonal decompositions in the case $p = 2$.

This is essentially the decomposition developed by Pease [40].

To obtain tridiagonal decompositions of the Fourier matrices of arbitrary size, methods based on identities other than the twiddle free and the general radix identities must be used. Towards this end we formulate a matrix identity associated with the Rader prime algorithm.

Let $\alpha \in \mathbf{Z}_p$ with $\alpha \neq 0, 1$. Let $R_{1,p}(\alpha)$ and $R_{2,p}(\alpha)$ be the permutation matrices corresponding to the permutations on \mathbf{Z}_p defined by:

$$\sigma_1: \begin{matrix} \alpha^i \rightarrow i \\ 0 \rightarrow 0 \end{matrix} \quad \sigma_2: \begin{matrix} i \rightarrow \alpha^{-i} \\ 0 \rightarrow 0 \end{matrix}.$$

Let $c_i \equiv \omega_p^{\alpha^{p-1-i}}$ and let C_p be the $(p-1) \times (p-1)$ circulant matrix $C_p \equiv \text{circ}(c_0, c_1, \dots, c_{p-2})$.

Let A be any $n \times n$ matrix. We denote by $A^{(m)}$ the $(n+m) \times (n+m)$ matrix

$$A^{(m)} = \begin{bmatrix} I_m & 0 \\ 0 & A \end{bmatrix}$$

We denote by U_p the $p \times p$ matrix with all one's in the first column, one's down the diagonal, and zeroes elsewhere. For example if $p = 5$, then

$$U_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, let \hat{F}_p denote the $p \times p$ matrix defined by $\hat{F}_p \equiv \text{blockdiag}[1, (\omega_p^{ij} - 1)]$ where $i, j \in [1, p-1]$. The matrix formulation of the Rader prime algorithm is then given in parts 1.) and 2.) of the following theorem.

Theorem 4.12. *Let $f_p(x)$ be the polynomial defined by*

$$f_p(x) \equiv c_0 + c_1 x + c_2 x^2 + \dots + c_{p-2} x^{p-2}$$

and Λ_p the $p \times p$ matrix

$$\Lambda_p \equiv \text{diag}(1, f_p(\omega_{p-1}^{i-1})), \quad i=1, 2, \dots, p-1.$$

Then

$$1.) F_p = U_p \hat{F}_p U_p^t$$

$$2.) \hat{F}_p = R_{1,p}(\alpha) C_p^{(1)} R_{2,p}(\alpha).$$

$$3.) C_p^{(1)} = F_{p-1}^{(1)} \Lambda_p F_p^{*(1)}$$

Proof.

The proof of 1.) consists of a transparent matrix multiplication and was observed by Parlett [39]. The identity 3.) follows from the matrix form of the convolution theorem (Section 3.2).

To prove 2.) we write

$$R_{1,p}(\alpha) = \left[\begin{array}{c|ccc} 1 & & & \\ \hline & & & \\ & & & \\ & & & \\ & & R & \end{array} \right] \quad \text{and} \quad R_{2,p}(\alpha) = \left[\begin{array}{c|ccc} 1 & & & \\ \hline & & & \\ & & & \\ & & & \\ & & S & \end{array} \right]$$

where R and S are the permutation matrices corresponding to the restrictions of σ_1 and σ_2 to the set $\{1, 2, \dots, p-1\}$ and the blank spaces represent zero entries. We also denote

$$\hat{F}_p = \left[\begin{array}{c|ccc} 1 & & & \\ \hline & & & \\ & & & \\ & & F & \end{array} \right]$$

where $F = (\omega_p^{ij} - 1)$. Since $R^{-1} = R^t$ and $S^{-1} = S^t$, it is sufficient to show that

$R^t F S^t = C_p$. If we denote $A = (a_{ij}) = R^t F S^t$ where $i, j \in [1, p-1]$, then $a_{ij} = \omega^{\sigma_1^{-1}(i)\sigma_2(j)} - 1$.

Note that if $j \in [1, p-1]$, then $\sigma_1^{-1}(1)\sigma_2(j) = \alpha\alpha^{-j} = \alpha^p\alpha^{-j}$ which implies that $a_{1j} = c_{j-1}$.

Thus, the first row of A is equal to the first row of C_p . The fact that A is circulant follows from the identity $\sigma_1^{-1}(i+k)\sigma_2(j+k) = \alpha^{i+k}\alpha^{-(j+k)} = \alpha^i\alpha^{-j} = \sigma_1^{-1}(i)\sigma_2(j)$. Therefore, $A = C_p$ and the theorem is proved.

Q.E.D.

Putting these identities together yields $F_p = U_p R_{1,p}(\alpha) F_{p-1}^{(1)} \Lambda_p F_{p-1}^{*(1)} R_{2,p}(\alpha) U_p^t$ where $*$ denotes the conjugate transpose.

Observe that if A and B are any $n \times n$ matrices, then $(I_m \otimes A^*) = (I_m \otimes A)^*$ and $A^{(m)}B^{(m)} = (AB)^{(m)}$. Therefore, if $F_{p-1} = \prod T_i$ is a tridiagonal decomposition of F_{p-1} , then $I_m \otimes F_{p-1}^{(1)} = I_m \otimes (\prod T_i)^{(1)} = I_m \otimes \prod (T_i^{(1)}) = \prod (I_m \otimes T_i^{(1)})$ is a tridiagonal decomposition of $I_m \otimes F_{p-1}^{(1)}$. Hence, if we can determine tridiagonal decompositions of U_p for p an odd prime, then we can proceed inductively to factor $F_{p-1}^{(1)}$ until $p = 2$ or $p = 3$. Thus, we need only determine methods for factoring the matrices U_p for p an odd prime to complete our description of the FFT-based decompositions.

Theorem 4.13. *Let p be an odd prime. Then*

$$U_p = \left(\prod_{k=p-1}^2 \left[\begin{array}{cccccc} I_k & 0 & 0 & & & \\ 0, \dots, 0, 1 & 1 & 0 & & & \\ 0 & 0 & I_{p-k-1} & & & \end{array} \right] \right) \begin{bmatrix} 1 & 0 & . & . & 0 & 0 \\ 1 & 1 & 0 & . & . & . \\ 0 & -1 & 1 & . & . & . \\ . & 0 & -1 & . & . & . \\ . & . & 0 & . & 0 & . \\ . & . & . & . & 1 & 0 \\ 0 & 0 & 0 & . & -1 & 1 \end{bmatrix}$$

Proof.

Let

$$U \equiv \left(\prod_{k=p-1}^2 \begin{bmatrix} I_k & 0 & 0 \\ 0, \dots, 0, 1 & 1 & 0 \\ 0 & 0 & I_{p-k-1} \end{bmatrix} \right) \begin{bmatrix} 1 & 0 & . & . & 0 & 0 \\ 1 & 1 & 0 & . & . & . \\ 0 & -1 & 1 & . & . & . \\ . & 0 & -1 & . & . & . \\ . & . & 0 & . & 0 & . \\ . & . & . & . & 1 & 0 \\ 0 & 0 & 0 & . & -1 & 1 \end{bmatrix} \equiv (u_{ij})$$

and denote

$$V_k \equiv \begin{bmatrix} I_k & 0 & 0 \\ 0, \dots, 0, 1 & 1 & 0 \\ 0 & 0 & I_{p-k-1} \end{bmatrix}$$

and

$$V \equiv \begin{bmatrix} 1 & 0 & . & . & 0 & 0 \\ 1 & 1 & 0 & . & . & . \\ 0 & -1 & 1 & . & . & . \\ . & 0 & -1 & . & . & . \\ . & . & 0 & . & 0 & . \\ . & . & . & . & 1 & 0 \\ 0 & 0 & 0 & . & -1 & 1 \end{bmatrix}.$$

Let $\mathbf{x} = (x_0, x_1, \dots, x_{p-1})^t \in \mathbb{C}^p$. We show that $U_p \mathbf{x} = U \mathbf{x}$. Note that

$$U_p \mathbf{x} = (x_0, x_1 + x_0, x_2 + x_0, \dots, x_{p-1} + x_0)^t.$$

Now

$$V \mathbf{x} = (x_0, x_0 + x_1, x_2 - x_1, x_3 - x_2, \dots, x_{p-1} - x_{p-2})^t.$$

Furthermore, if $\mathbf{y} = (y_0, y_1, \dots, y_{p-1})^t \in \mathbb{C}^p$, then

$$V_k \mathbf{y} = (y_0, y_1, \dots, y_{k-1}, y_k + y_{k-1}, y_{k+1}, \dots, y_{p-1})^t.$$

Thus,

$$V_2(V \mathbf{x}) = (x_0, x_0 + x_1, x_0 + x_2, x_3 - x_2, \dots, x_{p-1} - x_{p-2})^t$$

and one can show by induction that

$$V_j(V_{j-1}V_{j-2}\cdots V_2V\mathbf{x}) = (x_0, x_0+x_1, \dots, x_0+x_j, x_{j+1}-x_j, \dots, x_{p-1}-x_{p-2})^t$$

for $j \in [2, p-1]$. Hence, we may conclude that $U\mathbf{x} = U_p\mathbf{x}$ for every $\mathbf{x} \in C^p$ which implies that $U = U_p$.

Q.E.D.

A graphical representation of the decompositions is given in Figure 4.1.

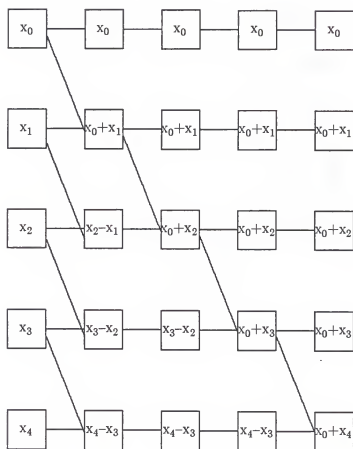


Figure 4.1. Local Computation of U_5 .

We have described how the problem of finding tridiagonal decompositions of Fourier matrices can be solved by applying matrix identities associated with the FFT and the Rader prime algorithm. Theorems 4.7-4.13 suggest the following algorithm for computing tridiagonal decompositions of F_n .

1.) Use either the general radix or twiddle free identity to factor F_n into a product of permutation matrices and block diagonal matrices with diagonal blocks of the form F_p for p a prime divisor of n .

2.) Use Theorem 4.12 to factor F_p for the odd prime divisors p of n .

3.) Use Theorem 4.13 to factor U_p and U_p^t .

4.) Go to 1.) for each $p-1$ with $p \neq 2, 3$.

5.) Continue until all the prime divisors are 2 or 3.

In the Appendix, we present computer programs based on these ideas which are written in an extended FORTRAN 77 which provides for the use of image algebra operations.

We briefly examine the effects of applying the identities directly to the two-dimensional Fourier matrices. One can use the resulting identities to map two-dimensional DFTs onto arrays which have dimensions smaller than the image dimensions or to reduce the number of parallel multiplication steps that are required by a factor of two when using the general radix identity.

Assume for simplicity that X is an $n \times n$ array and that $n = mk$. Using the general radix identity we can write

$$F_{n \times n} = F_n \otimes F_n =$$

$$\begin{aligned}
&= \left[(F_m \otimes I_k) D(k, m) (I_m \otimes F_k) P(k, m) \right] \otimes \left[(F_m \otimes I_k) D(k, m) (I_m \otimes F_k) P(k, m) \right] = \\
&= P_1 (I_{k^2} \otimes (F_m \otimes F_m)) P_2 (D(k, m) \otimes D(k, m)) P_3 (I_m \otimes (F_k \otimes F_k)) P_4
\end{aligned}$$

where P_1, P_2, P_3 , and P_4 are permutation matrices. This identity expresses $F_{n \times n}$ as a product of diagonal matrices, permutation matrices, and two-dimensional DFTs of smaller order. This can be used to map an $n \times n$ DFT onto a smaller array. It would not lead to a good algorithm in the event that the array was big enough for the image due to the increased number of permutations. One can rewrite the equation as follows:

$$\begin{aligned}
F_{n \times n} &= ((F_m \otimes I_k) \otimes I_n) (I_n \otimes (F_m \otimes I_k)) (D(k, m) \otimes D(k, m)) \\
&\quad ((I_m \otimes F_k) \otimes I_n) (I_n \otimes (I_m \otimes F_k)) (P(m, k) \otimes P(m, k)).
\end{aligned}$$

This decomposition still compresses two multiplication steps into one in the matrix $(D(k, m) \otimes D(k, m))$ but uses the separability properties of the Fourier matrices to keep the problem on a one-dimensional level with respect to the other terms in the expression. This equation represents the best of both worlds, in a sense; the number of multiplication steps is decreased and the number of permutation steps is held fixed. Therefore, using this identity along with the decompositions developed in this chapter would seem to be the best path to follow for implementation on a mesh-connected array.

4.2. Complexity of the FFT-Based Method

In this section we derive upper bounds on the number of parallel steps required to implement the algorithms implied by the decompositions derived in the previous section. We consider a basic step to be the equivalent of one \oplus operation with a local template. Thus, given a tridiagonal decomposition of a matrix, we consider the parallel complexity to be the number of tridiagonal matrices appearing in the decomposition plus the number of parallel steps required to implement the permutations. Due to the structure of the FFT-based decompositions, the complexity can be divided into multiplication,

addition, and permutation terms. We briefly consider the parallel multiplicative and additive complexity of the algorithms implied by the previous decompositions. Since these decompositions are related to FFTs, the associated complexity counts are similar to those of FFTs. The complexity of the permutations is then considered. It will be seen that it is the latter that dominates the overall complexity in most cases in terms of total number of steps. This can be attributed to the fact that FFTs are based on reindexing schemes. When implementing an algorithm on an architecture such as mesh-connected arrays, reindexing can be an “expensive” operation. A local permutation step is much less expensive than a floating point multiplication however. We parameterize the times required to implement a local multiplication step, a local addition step, and a local permutation step based on execution times for the Massively Parallel Processor to compare the general radix and twiddle free identities [60].

For any matrix A we denote by $C_m(A)$ and $C_a(A)$ the number of parallel multiplication and addition steps required to implement A locally on a linear array. By a parallel permutation step we mean the action of switching the data in horizontally or vertically adjacent cells. We denote by $C_r(A)$ the number of parallel permutation, or data routing, steps required to implement A locally. We acknowledge that there is some ambiguity in this notation since there can be more than one decomposition available for a given A and the complexity measures with respect to the different decompositions may be different. It will either be explicitly stated which decomposition is being used or it will be clear from the context.

Remark 4.14. Note that if $x = m, a,$ or r , then

$$C_x(AB) = C_x(A) + C_x(B)$$

and, since we are considering *parallel* complexity,

$$C_x(I_j \otimes A) = C_x(A)$$

for any matrices A and B and positive integer j.

We first consider the number of parallel multiplication steps. We suppress the multiplication by $n^{-1/2}$.

Theorem 4.15. *Let n be as in Theorem 4.7.*

1.) *Using Corollary 4.8 followed by Corollary 4.11 to decompose F_n results in*

$$C_m(F_n) = \left[\sum_{j=1}^s (k_j C_m(F_{p_j}) + k_j) \right] - s.$$

2.) *Using Corollary 4.10 followed by Corollary 4.11 to decompose F_n results in*

$$C_m(F_n) = \left[\sum_{j=1}^s (k_j C_m(F_{p_j}) + k_j) \right] - 1.$$

where we allow s to be equal to 1.

3.) *If p is an odd prime, then using Theorem 4.12 results in*

$$C_m(F_p) = 2C_m(F_{p-1}) + 1.$$

Proof.

1.) By Corollary 4.8,

$$\begin{aligned} C_m(F_n) &= \left[\sum_{j=0}^{s-2} C_m(E_j) \right] + C_m(I_{n_{p-1}} \otimes F_{q_s}) = \\ &= \left[\sum_{j=0}^{s-2} C_m(F_{q_{j+1}}) \right] + C_m(F_{q_s}) = \sum_{j=1}^s C_m(F_{p_j k_j}). \end{aligned}$$

By Corollary 4.11, if p is any prime and $k > 1$

$$\begin{aligned}
C_m(F_{p^k}) &= \sum_{m=0}^{k-2} \left[C_m(I_{p^{k-1}} \otimes F_p) + C_m(I_{p^m} \otimes D_{p^{k-m}}) \right] + C_m(F_p) = \\
&= \sum_{m=0}^{k-2} C_m(F_p) + k-1 + C_m(F_p) = k C_m(F_p) + k-1.
\end{aligned}$$

Hence,

$$C_m(F_n) = \left[\sum_{j=1}^s (k_j C_m(F_{p_j}) + k_j) \right] - s$$

which proves 1.).

2.) The only difference here (in terms of multiplications) is the presence of the twiddle factors. Hence,

$$\begin{aligned}
C_m(F_n) &= \left[\sum_{j=1}^s C_m(F_{p_j}^{k_j}) \right] + s-1 \\
&= \left[\sum_{j=1}^s (k_j C_m(F_{p_j}) + k_j) \right] - 1.
\end{aligned}$$

3.) This follows immediately from Theorem 4.12.

Q.E.D.

Hence, the difference in the number of parallel multiplications required by the two different methods is $s-1$ multiplication steps. Furthermore, as pointed out in the previous section, on a two-dimensional array the difference can be kept fixed at $s-1$ multiplication steps without increasing the number of steps required to execute the other operations. Note that $C_m(F_2) = 0$ and that if $n = 2^k$, then $C_m(n) = \log_2 n - 1$.

Theorem 4.16. *Using either method results in the following expressions for the additive complexities:*

1.) If $n = \prod_{j=1}^s p_j^{k_j}$ is not prime, then

$$C_a(F_n) = \sum_{j=1}^s k_j (C_a(F_{p_j})).$$

2.) If p is an odd prime, then

$$C_a(F_p) = 2C_a(F_{p-1}) + 2(p-1).$$

Proof.

The proof of 1.) is almost identical to the proofs of the previous theorem. The only difference is that the twiddle factors require no additions. The proof of 2.) follows immediately from Theorems 4.12 and 4.13.

Q.E.D.

Note that the additions due to the matrices U_p will have a significant influence on the additive complexity. This is because the method of computing the U_p , although local, is, except for one step, essentially serial.

We now move to a discussion of the permutation complexity. The permutations can be executed locally by the odd-even transposition sort (OETS) which is a parallel algorithm for sorting data arranged in a one-dimensional array.

Remark 4.17. It has been shown that the OETS will execute an arbitrary permutation of n objects on a linear array in at most n parallel steps [28, 62]. We will show that, because of the special structure of the shuffle permutations, the OETS will execute the permutation σ_{mk} corresponding to $P(m, k)$ in either $(m-1)(k-1)$ or $(m-1)(k-1)+1$ parallel steps. In order to keep the complexity estimates together, we delay proving this result until after we have derived upper bounds on the parallel permutation complexities.

In what follows, we make the convention that $\sum_{j=m}^n x_j \equiv 0$ if $n < m$.

Theorem 4.18. *Upper bounds on the number of parallel permutation steps required to implement the various algorithms are:*

1.) *If n is as in Theorem 4.7 and Corollary 4.8 is used to decompose F_n , then*

$$C_r(F_n) \leq n + \sum_{j=0}^{s-2} c_{j-1} + (p_{s-1}^{k_{s-1}-1})(p_s^{k_s-1}) + 1 + \sum_{j=1}^s C_r(F_{p_j^{k_j}}).$$

2.) *If n is as in Theorem 4.7 and Corollary 4.10 is used to decompose F_n , then*

$$C_r(F_n) \leq 3n - 2 \sum_{j=1}^s p_j^{k_j} + 4(s-1) + \sum_{j=1}^s C_r(F_{p_j^{k_j}}).$$

3.) *If $n = p^k$ for $k \geq 2$ and Corollary 4.11 is used to decompose F_n , then*

$$C_r(F_n) \leq 3n - 2pk + kC_r(F_p).$$

4.) *If $n = p$ is an odd prime and Theorem 4.12 is used to decompose F_n , then*

$$C_r(F_n) \leq 2(n-1) + 2C_r(F_{n-1}).$$

Thus, in any case, the upper bounds on the number of parallel permutation steps is linear in n , that is, $C_r(F_n)$ is $o(n)$.

Proof

1.) Denote $R \equiv \sum_{j=1}^s C_r(F_{p_j^{k_j}})$. Since $c_{s-1} = p_s^{k_s}$,

$$C_r(F_n) \leq \left[\sum_{j=0}^{s-2} C_r(R_j) \right] + C_r(P(p_{s-1}^{k_{s-1}}, p_s^{k_s})) + R + C_r(\left(\prod_{j=1}^{s-1} (I_{n_{p-j-1}} \otimes Q_{2(s-j)}) \right)).$$

By definition of R_j and Remark 4.14,

$$C_r(F_n) \leq C_r(P(q_j, c_j)(I_{q_j} \otimes Q_{2j+1})(I_{q_j} \otimes P(c_{j+1}, q_{j+1}))) \leq q_j c_j = c_{j-1}.$$

Since $P(p_{s-1}^{k_{s-1}}, p_s^{k_s})$ is a shuffle permutation, $C_r(P(p_{s-1}^{k_{s-1}}, p_s^{k_s})) \leq (p_{s-1}^{k_{s-1}-1})(p_s^{k_s-1}) + 1$.

The permutation $Q = \prod_{j=1}^{s-1} (I_{n_{p-j-1}} \otimes Q_{2(s-j)})$ can be implemented as a sequence of per-

mutations or as a one step permutation. Note that

$$Q_2 = T_{q_1}(C_{c_1}^{q_1^*})P(c_1, q_1)$$

so

$$C_r(Q_2) \leq c_1 + (c_1 - 1)(q_1 - 1) + 1 = n - q_1 + 2 = c_0 - q_1 + 2.$$

Similarly

$$Q_{2(s-j)} = T_{q_{s-j}}(C_{c_{s-j}}^{q_{s-j}^*})P(c_{s-j}, q_{s-j})$$

so

$$C_r(Q_{2(s-j)}) \leq c_{s-j} + (c_{s-j} - 1)(q_{s-j} - 1) + 1 = c_{s-j-1} - q_{s-j} + 2.$$

Thus, implementing Q as a sequence of permutations leads to

$$C_r(Q) \leq \sum_{j=1}^{s-1} c_{j-1} = n + \frac{n}{n_1} + \frac{n}{n_2} + \cdots + \frac{n}{n_{s-2}} > n.$$

Implementing Q as a one step permutation yields $C_r(Q) \leq n$.

Combining the observations made in the previous two paragraphs yields

$$C_r(F_n) \leq C_r(Q) + \sum_{j=0}^{s-2} c_{j-1} + (p_{s-1}^{k_{s-1}-1})(p_s^{k_s-1}) + 1 + R$$

which proves 1.).

2.) Let R be as in 1.). By Corollary 4.10,

$$C_r(F_n) \leq 2 \sum_{j=1}^{s-1} C_r(P_{c_{j-1}}) + C_r\left(\prod_{j=2}^s (I_{n_{s-j}} \otimes P_{c_{s-j}})\right) + R.$$

Since the $P_{c_{j-1}} = P(c_j, p_j^{k_j})$ are shuffle permutations, $C_r(P_{c_{j-1}}) \leq (c_j - 1)(p_j^{k_j} - 1) + 1$. Note

that $c_j p_j^{k_j} = c_{j-1}$. Hence,

$$\sum_{j=1}^{s-1} C_r(P_{c_{j-1}}) \leq \sum_{j=1}^{s-1} c_{j-1} - \sum_{j=1}^{s-1} c_j - \sum_{j=1}^{s-1} p_j^{k_j} + 2(s-1) =$$

$$\begin{aligned}
&= c_0 - c_{s-1} - \sum_{j=1}^{s-1} p_j^{k_j} + 2(s-1) = \\
&= n - p_s^{k_s} - \sum_{j=1}^{s-1} p_j^{k_j} + 2(s-1) = \\
&= n - \sum_{j=1}^s p_j^{k_j} + 2(s-1).
\end{aligned}$$

By similar reasoning as used in 1.) for the permutation matrix Q ,

$$C_r(\prod_{j=2}^s (I_{n_{s-j}} \otimes P_{c_{s-j}})) \leq n.$$

Thus,

$$C_r(F_n) \leq 2n + 4(s-1) - 2 \sum_{j=1}^{s-1} p_j^{k_j} + n + R$$

which proves 2.).

3.) By Corollary 4.11

$$\begin{aligned}
C_r(F_n) &= C_r(G_p(I_{p^{k-1}} \otimes F_p)H_p) \\
&= 2 \sum_{m=0}^{k-2} C_r(P(p^{k-m-1}, p)) + \sum_{m=0}^{k-1} C_r(F_p) + C_r(H_p) \leq \\
&\leq 2 \sum_{m=0}^{k-2} \left[(p^{k-m-1}-1)(p-1)+1 \right] + kC_r(F_p) + n.
\end{aligned}$$

A little algebra yields

$$\begin{aligned}
&\sum_{m=0}^{k-2} \left[(p^{k-m-1}-1)(p-1)+1 \right] = \\
&= p^{k-1}(p-1) \left(\sum_{m=0}^{k-2} \left(\frac{1}{p} \right)^m \right) - (k-1)(p-1) + (k-1) = \\
&= p^{k-1}(p-1) \left(\frac{p^{k-1}-1}{p^{k-1}} \frac{p}{p-1} \right) - p(k-1) = \\
&= p(p^{k-1}-1) - p(k-1) = n - pk.
\end{aligned}$$

Therefore,

$$C_r(F_n) \leq 3n - 2pk + kC_r(F_p)$$

which proves 3.).

4.) This inequality follows immediately from Theorem 4.12.

Q.E.D.

The upper bounds on the permutation complexities can be sharpened in certain special cases. This is because the structure of the permutations is not destroyed (or clouded over) by the presence of too many permutation matrices.

Theorem 4.19. *If n is as in Theorem 4.7 with $s = 2$ then*

1.) *If the twiddle free identity is used to decompose F_n , then*

$$C_r(F_n) \leq 3(n+6) - 2(p_2^{k_2} + p_1^{k_1}) + C_r(F_{p_1^{k_1}}) + C_r(F_{p_2^{k_2}}).$$

2.) *If the general radix identity is used to decompose F_n , then*

$$C_r(F_n) \leq 3(n+2) - 3(p_1^{k_1} + p_2^{k_2}) + C_r(F_{p_1^{k_1}}) + C_r(F_{p_2^{k_2}}).$$

3.) *If $n = p^2$, then*

$$C_r(F_n) \leq 3(p-1)^2 + 3 + 2C_r(F_p).$$

Proof.

1.) Let $m = p_1^{k_1}$ and $k = p_2^{k_2}$. The twiddle free identity can be written

$$F_n = P(k, m) T_k(C_m^{-k^*}) (I_k \otimes F_m) P(m, k) (I_m \otimes F_k) T_m(C_k^{m^*}) P(k, m).$$

Since $T_m(C_k^{m^*})$ is a block diagonal permutation matrix with $k \times k$ blocks,

$$C_r(T_m(C_k^{m^*})) \leq k. \text{ Hence,}$$

$$C_r(T_m(C_k^{m^*})P(k,m)) \leq (k-1)(m-1)+1+k = n-m+2.$$

Similarly,

$$C_r(P(k,m)T_k(C_m^{k^*})) \leq (k-1)(m-1)+1+m = n-k+2.$$

Whence,

$$\begin{aligned} C_r(F_n) &\leq n-k+2 + (m-1)(k-1)+1 + n-m+2 + C_r(F_m) + C_r(F_k) = \\ &= 3n + 6 - k - 2m + C_r(F_m) + C_r(F_k) \end{aligned}$$

which proves 1.).

2.) Let m and k be as in 1.). The general radix identity is

$$F_n = P(k,m)(I_k \otimes F_m)P(m,k)D(k,m)(I_m \otimes F_k)P(k,m).$$

Thus,

$$\begin{aligned} C_r(F_n) &= 3C_r(P(k,m)) + C_r(F_m) + C_r(F_k) \leq \\ &\leq 3(m-1)(k-1)+3 + C_r(F_m) + C_r(F_k) = \\ &= 3n - 3(m+k) + 6 + C_r(F_m) + C_r(F_k) \end{aligned}$$

which proves 2.).

3.) In this case the general radix identity is

$$F_n = P(p,p)(I_p \otimes F_p)P(p,p)D(p,p)(I_p \otimes F_p)P(p,p)$$

from which the desired inequality follows immediately.

Q.E.D.

Remark 4.20. Note that the complexity bound corresponding to the twiddle free identity in Theorem 4.18 is not symmetric in the prime factors of n . This is due to the fact that a better estimate can be obtained for $P(p_{s-1}^{k_{s-1}}, p_s^{k_s})$ than for $(P(q_j, c_j)(I_{q_j} \otimes Q_{2j+1})(I_{q_j} \otimes P(c_{j+1}, Q_{j+1})))$. Thus, if using this identity it would be wise to order the prime factors so that the quantity $(p_1^{k_1} - 1)(p_2^{k_2} - 1)$ is minimized.

Remark 4.21. The difference in the bounds in 1.) and 2.) of Theorem 4.18 is the quantity

$$D = \sum_{j=0}^{s-1} c_{j-1} - 2n + (p_1^{k_1} - 1)(p_2^{k_2} - 1) + 2 \sum_{j=1}^s p_j^{k_j} + 1 + 4(s-1).$$

Since $c_{-1} = n$, $p_1^{k_1} p_2^{k_2} \leq n$, and $2 \sum_{j=1}^s p_j^{k_j} - p_1^{k_1} - p_2^{k_2} \geq 0$, it follows that $D > 0$. Thus, as

expected, it takes more permutation steps using the twiddle free identity. In fact, let P_0 denote the time it takes to execute one parallel permutation step and M_0 the time it takes to execute one multiplication step. Then, disregarding the overhead involved in implementing the various operations, for the general radix identity to be preferable to

the twiddle free identity we should have $(s-1)M_0 < DP_0$ or $R = \frac{M_0}{P_0} < \frac{D}{s-1}$. In the

special cases given in Theorem 4.19 the difference in the bounds is $\bar{D} = p_1^{k_1} + p_2^{k_2}$. On the Massively Parallel Processor the quantity R is on the order of 10^3 [60]. Since the GAPP has the same clock rate as the MPP it is reasonable to expect that the quantity is about the same for the GAPP [54]. What we are neglecting here is the fact that the physical actions of moving the data or performing the multiplications need to be controlled. The instructions need to be broadcast to the individual processors. Moreover, when implementing the permutations comparisons need to be made in order that a decision can be made to move the data or not. These things take time and are valid considerations when attempting to compare and evaluate the algorithms resulting from the decompositions. An evaluation of the effects of such considerations requires a knowledge of the basic instructions of the processors, which we do not have.

We have computed some of the quantities of interest related to the complexities of the various decompositions and used them to construct Table 4.1.

Table 4.1. Estimated number of parallel steps required to implement F_n locally.

n	C_r :GR	C_r :TF	C_m :GR	C_m :TF	$C_a(F_n)$
100 (25·4)	227	316	9	8	26
120 (8·5·3)	367	400	8	6	21
128	356	—	6	—	7
256	736	—	7	—	8
360 (9·8·5)	1096	1161	11	9	27
500(125·4)	1506	1648	21	20	38
512	1500	—	8	—	9

GR denotes use of the general radix identity. TF denotes use of the twiddle free identity. C_r denotes permutation steps. C_m denotes multiplication steps. C_a denotes addition steps.

We now show that the shuffle permutations corresponding to $P(m,n)$ are executed by the odd-even transposition sort (OETS) in either $(n-1)(m-1)$ or $(n-1)(m-1)+1$ steps. The OETS is a method of executing a permutation of a linear array of data using a sequence of adjacent transpositions. The idea is to apply the same sequence of transpositions to the data that is required to sort the vector

$$(\sigma(0), \sigma(1), \dots, \sigma(n-1))$$

into increasing order, where σ is the permutation to be implemented.

We first formally define the OETS and give an example of it. We then establish some notation and prove a preparatory lemma before proving the aforementioned result.

Let σ be any permutation on Z_{mn} and define a function $\vec{h}_\sigma \equiv \vec{h} : \{0,1,2,\dots\} \rightarrow Z_{mn}^{mn}$ by defining

$$\vec{h}(0) \equiv (\sigma(0), \sigma(1), \dots, \sigma(mn-1))$$

and, for each $k \in [1, \infty)$, defining $\vec{h}(k)$ recursively as follows:

1.) If k is odd, then $\vec{h}(k) = (h_{k,0}, h_{k,1}, \dots, h_{k,mn-1})$

where

$$h_{k,j} = \begin{cases} h_{k-1,j-1} & \text{if } j \text{ is odd and } h_{k-1,j} < h_{k-1,j-1} \\ h_{k-1,j+1} & \text{if } j \text{ is even, } j < mn-1, \text{ and } h_{k-1,j} > h_{k-1,j+1} \\ h_{k-1,j} & \text{else} \end{cases}$$

2.) If k is even, then $\vec{h}(k) = (h_{k,0}, h_{k,1}, \dots, h_{k,mn-1})$

where

$$h_{k,j} = \begin{cases} h_{k-1,j-1} & \text{if } j \text{ is even, } j \neq 0, \text{ and } h_{k-1,j} < h_{k-1,j-1} \\ h_{k-1,j+1} & \text{if } j \text{ is odd, and } h_{k-1,j} > h_{k-1,j+1} \\ h_{k-1,j} & \text{else} \end{cases}$$

As mentioned earlier, it has been shown that if σ is any permutation on \mathbf{Z}_{mn} , then $\vec{h}(mn-1) = (0, 1, 2, \dots, mn-1)$ and that if $k > mn-1$, then $\vec{h}(k) = \vec{h}(mn-1)$. The algorithm used to generate the vectors $\vec{h}(k)$ from $\vec{h}(0)$ using steps 1.) and 2.) is called the *odd-even transposition sort*. The fact that $\vec{h}(mn-1)$ is arranged in increasing order means that the coordinates of $\vec{h}(0)$ have been permuted by σ . The following example shows how the OETS works.

Let $\sigma = \sigma_{3,4}$ be the shuffle permutation corresponding to the mapping $a+3b \rightarrow 4a+b$. Then

$$\vec{h}(0) = (0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11)$$

$$\vec{h}(1) = (0, 4, 1, 8, 5, 9, 2, 6, 3, 10, 7, 11)$$

$$\vec{h}(2) = (0, 1, 4, 5, 8, 2, 9, 3, 6, 7, 10, 11)$$

$$\vec{h}(3) = (0, 1, 4, 5, 2, 8, 3, 9, 6, 7, 10, 11)$$

$$\vec{h}(4) = (0,1,4,2,5,3,8,6,9,7,10,11)$$

$$\vec{h}(5) = (0,1,2,4,3,5,6,8,7,9,10,11)$$

$$\vec{h}(6) = (0,1,2,3,4,5,6,7,8,9,10,11)$$

To permute a one-dimensional array of data $(a(0),a(1),\dots,a(11))$ locally using the OETS one uses the sequence of transpositions used to generate $\vec{h}(6)$ from $\vec{h}(0)$. Note that the algorithm stabilizes in $(3-1)(4-1)$ steps.

We now establish notation. Let $n, m > 1$ be arbitrary but fixed positive integers and let $\vec{h} = \vec{h}_{\sigma_{mn}}$. Let $p_x(k)$ denote the position of $x \in \mathbf{Z}_{mn}$ at step k , that is, if $x = h_{k,j}$ then $p_x(k) = j$. Let $b_x(k)$ denote the number of elements $y \in \mathbf{Z}_{mn}$ with the property that $p_y(k) < p_x(k)$ and $y > x$. Similarly, let $\bar{b}_x(k)$ be the number of elements $y \in \mathbf{Z}_{mn}$ with the property that $p_y(k) > p_x(k)$ and $y < x$. Note that if $b_x(k) = 0$ for every $x \in \mathbf{Z}_{mn}$, then $\vec{h}(k)$ is sorted in increasing order. Note further that if $b_x(k) = 0$, then $b_x(j) = 0$ for every $j \geq k$. These observations hold true for \bar{b}_x also. The criteria that we shall use to determine if \vec{h} is sorted is given in the next lemma.

Lemma 4.23. *If $\mathbf{Z}_{mn} = H \cup K$ such that for some positive integer k , $x \in H$ and $y \in K$ implies that $x < y$ and $b_x(k) = \bar{b}_y(k) = 0$, then $\vec{h}(k)$ is sorted.*

Proof.

Assume that the lemma is not true. Then there exists $z \in \mathbf{Z}_{mn}$ such that $p_z(k) \neq z$. In fact, we can take z such that $p_z(k) > z$. There must be a $w \in \mathbf{Z}_{mn}$ such that $w > z$ and $p_w(k) < p_z(k)$ since there are more than z positions to fill to the left of z . Hence $b_z(k) \neq 0$ which implies that $z \notin H$ so $z \in K$. The same inequalities imply that $\bar{b}_w(k) \neq 0$. Thus, $w \notin K$ so $w \in H$. By hypothesis, this implies that $w < z$ which contradicts $w > z$. Therefore, we conclude that the lemma is true.

Before proving the Theorem 4.24, we point out that, although the proof is long, it is not hard to get an intuitive feel for why the OETS works so well for the shuffle permutations. By writing out a few of the permutations the reader can see that they are analogous to dealing a hand of cards. The objects are sorted in a uniform, shuffled fashion. Therefore, after the first few steps, many exchanges are made at each step. For those that prefer computational evidence to proofs, a computer program was written which executes $P(m,n)$ using the OETS. This program was run on all pairs $m,n \in [2,100]$ and Theorem 4.24 was verified in every case.

Theorem 4.24. *The odd-even transposition sort will execute $P(m,n)$ in either $(n-1)(m-1)$ or $(n-1)(m-1)+1$ parallel steps.*

Proof.

The proof is divided into two cases: m even and m odd.

Case 1. m even.

In this case we can write

$$\vec{h}(0) = (E_0^{(1)}, E_0^{(2)}, E_1^{(1)}, E_1^{(2)}, \dots, E_{n-1}^{(1)}, E_{n-1}^{(2)})$$

where

$$E_i^{(1)} = (i, n+i, 2n+i, \dots, (\frac{m}{2}-1)n+i)$$

$$E_i^{(2)} = (\frac{m}{2}n+i, (\frac{m}{2}+1)n+i, \dots, (m-1)n+i).$$

By abuse of notation, we write $x \in E_i^{(j)}$ to denote that x is one of the entries of the vector $E_i^{(j)}$.

Let $i \in [0, n-1]$. We show that if $x \in E_i^{(1)}$, then $b_x(1+i(m-1)) = 0$. Thus, in particular, it will follow that if $x \in E_i^{(j)}$ for any $j \in [0, n-1]$, then $b_x(1+(n-1)(m-1)) = 0$.

Note that if $x \in E_0^{(j)}$, then $b_x(0) = 0$. Let $i \in [1, n-1]$ and $j \in [0, \frac{m}{2}-1]$. Let $x = jn+i$ and let k_x be the smallest integer such that $b_x(k_x) = 0$. Note that due to the restrictions on i and j , $x \in E_i^{(1)}$. We show that the following claims are true:

Claim 1.) If $k \in [0, j+1]$, then $p_x(k) = p_x(0)$.

Claim 2.) If $k \in [j+2, k_x]$, then $p_x(k) = p_x(k-1) - 1$.

Note that Claims 1.) and 2.) together imply that $k_x = j + 1 + b_x(0)$.

Since m is even, the comparisons made at the first step of the algorithm are all made within the vectors $E_s^{(t)}$ for $s, t \in [0, n-1]$. These vectors are already sorted in increasing order. Thus, there are no exchanges made on the first pass of the algorithm. Claim 1.) then follows from the fact that if $y \in E_i^{(1)}$, $z \in E_{i-1}^{(2)}$, and $w \in E_i^{(2)}$, then $z > y$ and $w > y$.

We prove Claim 2.) by double induction on i and j . Let $i = 1$ and $j = 0$. The claim is then that $p_1(k) = p_1(k-1) - 1$ for $k \in [2, k_x]$. By Claim 1.), $p_1(1) = p_1(0)$. Since $p_0(0) = 0$, the inequality $0 < p_x(k) < p_1(k)$ implies that $z > 1$. Thus, $p_1(k) = p_1(k-1) - 1$. Assume that the claim is true for every $x = kn+1$ with $k \in [0, j-1]$ and $j \in [0, \frac{m}{2}-1]$. By Claim 1.),

$$p_{(j-1)n+1}(j+2) < p_{(j-1)n+1}(j+1) < p_{(j-1)n+1}(j) = p_{(j-1)n+1}(j-1) = \cdots = p_{(j-1)n+1}(0)$$

and

$$p_{jn+1}(j+2) < p_{jn+1}(j+1) = p_{jn+1}(j) = p_{jn+1}(j-1) = \cdots = p_{jn+1}(0).$$

Note that if $z \in E_1^{(1)}$ and $y \in E_0^{(2)}$, then $z > y$. Furthermore, if $z \in E_0^{(1)}$ and $z > x$,

then $z > (j-1)n+1 = x-n$ which implies that $k_{(j-1)n+1} \geq k_x$. Putting these last three observations together yields the equality $p_{j+1}(k) - p_{(j-1)n+1}(k) = 2$ for $k \in [j+2, k_x] \subset [j+2, k_{(j-1)n+1}]$. By the induction hypothesis on j , $p_{(j-1)n+1}(k) = p_{(j-1)n+1}(k-1) - 1$ for $k \in [j+2, k_x] \subset [j+1, k_{(j-1)n+1}]$. Thus, $p_{j+1}(k) = p_{j+1}(k) - 1$ for $k \in [j+2, k_x]$.

Assume that for some $i \in [2, n-1]$, Claim 2.) is true for every j_0n+i_0 with $i_0 \in [1, i-1]$ and $j_0 \in [0, \frac{m}{2}-1]$. Let $x = i$ and note that $\{j : j < i \text{ and } p_j(0) < p_i(0)\} = \{0, 1, \dots, i-1\}$. Thus $b_i(0) = i(m-1)$. By the induction hypothesis on i , $p_{i-1}(k) = p_{i-1}(k-1) - 1$ for $k \in [2, k_{i-1}]$. Moreover, $k_{i-1} = 0 + 1 + (i-1)(m-1)$ which yields $k_i \geq b_x(0) = i(m-1) \geq i(m-1) + 2 - m = k_{i-1}$. Assume that $k \in [2, k_i]$, $p_x(k) = p_i(k) - 1$, and $z < i$. Then $z = i-1$ so $b_{i-1}(k) = 0$. Therefore, $b_i(k) = 0$ since if $p_w(k) < p_i(k)$, and $w \neq i-1$, then $p_w(k) < p_{i-1}(k)$. Hence, $k = k_i$. This implies that for every $k \in [2, k_i]$, $p_{i-1}(k) = p_{i-1}(k-1) - 1$.

Assume that for some $j \in [1, \frac{m}{2}-1]$, Claim 2.) is true for every $hn+i$ with $h \in [1, j-1]$ and that it is not true for $x = jn+i$. Let k_0 be the smallest of all the integers k in the interval $[j+2, k_x]$ with the property that $p_x(k) = p_x(k-1)$. Note that if $k_0 = k_x$, then $b_x(k_x-1) = 0$ which contradicts the choice of k_x . Furthermore, $k_0 = j+2$ would imply that $p_x(j+1) = p_x(j+2)$ which is false by Claim 1.).

If $p_x(k_0-1) = p_x(k_0-1) - 1$, then, by our choice of k_0 , $z < x$. For such a z , it must be true that $p_x(k_0) = p_x(k_0-1)$ so $p_x(k_0) = p_x(k_0-1)$. By Claim 1.), $p_x(k) = p_x(0)$ for $k \in [0, j+1]$. By the induction hypothesis, if $y = sn+i$ with $s < j$, then $p_y(j+1) < p_y(0)$. Furthermore, if $z \in E_{i-1}^{(2)}$, then $x < z$ and if $z = sn+i-1 \in E_{i-1}^{(1)}$ and $s > j$, then $z > x$. Thus, if $p_z(j+1) = p_x(j+1) - 1$, then $z < x$ so $p_x(j+2) = p_x(j+1) - 1$. Hence, there exists a k_1 with $k_1 < k_0 < k_x$ and $p_z(k_1) = p_z(k_1-1)$, that is, x has to catch up to z . By

the induction hypothesis on i , this implies that $b_x(k_1) = 0$. Since $k_1 < k_0 < k_x$, $b_x(k_0) \neq 0$. Since $p_x(k_0) = p_x(k_1) - 1$ and $x > z$, it must be true that $b_x(k_0) \neq 0$ which implies that $b_x(k_1) \neq 0$ which is a contradiction. This proves Claim 2.)

We now need only show that given $x = i+j$, $k_x \leq 1+i(m-1)$. Recall that $k_x = j+1 + b_x(0)$. By definition, $x < z$ for every $z \in E_k^{(2)}$, $k \in [0, n-1]$, and $|E_k^{(2)}| = \frac{m}{2}$. Since x is the j^{th} element of $E_i^{(1)}$, x is less than $\frac{m}{2} - (j+1)$ of the elements of $E_k^{(1)}$ for $k < i$. Hence $b_x(0) = i\frac{m}{2} + i(\frac{m}{2} - (j+1)) = i(m - (j+1))$. Thus, $k_x = im - (j+1)(i-1)$ so the inequality will be true if and only if $(1-i)(j+1) \leq 1-i$ for $i \in [1, n-1]$ and $j \in [0, \frac{m}{2}-1]$. If $i = 1$, then the inequality is clearly satisfied. If $i > 1$, then the inequality is equivalent to $j+1 \geq 1$ which is also clearly true. Thus, we may conclude that if $x \in E_k^{(1)}$ for some k , then $b_x(1+(n-1)(m-1)) = 0$.

By replacing $<$'s by $>$'s and b 's by \bar{b} 's the same argument can be used to show that if $x \in E_k^{(2)}$ for some k , then $\bar{b}_x(1+(n-1)(m-1)) = 0$. Thus, by Lemma 4.23, the theorem is proved in the case m even.

Case 2: m odd.

In this case we write

$$\bar{h}(0) = (E_0^{(1)}, m_0, E_0^{(2)}, E_1^{(1)}, m_1, E_1^{(2)}, \dots, E_{n-1}^{(1)}, m_{n-1}, E_{n-1}^{(2)})$$

where

$$E_i^{(1)} = (i, n+i, 2n+i, \dots, (\frac{m-3}{2})n+i)$$

$$m_i = (\frac{m-1}{2})n+i$$

$$E_i^{(2)} = ((\frac{m+1}{2})n+i, (\frac{m}{2}+1)n+i, \dots, (m-1)n+i).$$

Since $m_i > x$ for every $x \in E_{i+1}^{(1)}$ and $m_i < x$ for every $x \in E_i^{(2)}$, the argument that was used in Case 1 can be used with very little modification to show that if $x \in E_i^{(1)}$ for any i , then $b_x(1+(n-1)(m-1)) = 0$ and if $x \in E_i^{(2)}$ for any i , then $\bar{b}_x(1+(n-1)(m-1)) = 0$. One modification to be made is in Claim 1.). Since m is odd, some exchanges take place during the first pass of the algorithm. This results in $k_x \leq j+1 + b_x(0)$ rather than $k_x = j+1 + b_x(0)$. One can then proceed with the same proof keeping in mind that for every $i \in [0, n-1]$, $x \in E_{i+1}^{(1)}$ and $y \in E_{i-1}^{(2)}$ imply that $x < m_i < y$ which allows for the steady movement of the data. After accounting for these facts, the inequality derived after the proof of Claim 2.) must be rederived, i.e., one needs to show that if $x = in+j$, then $k_x \leq 1+i(m-1)$. This can be accomplished by substituting $m-1$ for m . Let $x = (\frac{m-3}{2})n + n-1$. Then, since $x = m_0-1 < m_1 < \dots < m_{n-1}$, if $b_x(k) = 0$, then $b_{m_i}(k) = 0$ for $i \in [0, n-1]$. Since $x \in E_{n-1}^{(1)}$, $b_x(1+(n-1)(m-1)) = 0$ so $b_{m_i}(1+(n-1)(m-1)) = 0$ for $i \in [0, n-1]$. Note that if $y \in E_i^{(2)}$ for any i , then $m_{n-1} < y$. Hence, Lemma 4.23 applies and we may conclude that the OETS takes at most $(n-1)(m-1)+1$ steps in the case that m is odd.

We conclude the proof by noting that, since $|p_{n-1}(0) - (n-1)| = (n-1)(m-1)$ and since there may be no change in the first step of the algorithm, the OETS takes at least $(n-1)(m-1)$ steps.

Q.E.D.

We have shown how matrix identities associated with the FFT can be used to develop tridiagonal decompositions of Fourier matrices and we have evaluated the resulting algorithms. We now derive a completely different method for computing tridiagonal decompositions of Fourier matrices.

CHAPTER 5

TRIDIAGONAL DECOMPOSITIONS OF FOURIER MATRICES BY OBLIQUE ELIMINATION

In this chapter, we show how a technique called oblique elimination can be used to develop alternative tridiagonal decompositions of the Fourier matrices. This method has advantages over the FFT based method. The major advantage is that there is relatively little data manipulation required to implement the algorithms implied by these decompositions. Another advantage is that the parallel arithmetic operation counts are all the same linear function of n for *every* n regardless of the compositeness of n . This is due to the fact that oblique elimination is a method based on numerical linear algebra rather than traditional discrete Fourier transform methods. A disadvantage is that the number of multiplication steps is higher than for the FFT-based method, particularly for values of n such as powers of two.

The parallel algorithms resulting from these decompositions can be used alone or in conjunction with the FFT-based method. Specifically, the FFT-based method could be used to break the computation into prime components. The method described in this chapter could then be used in place of the Rader prime algorithm and the convolution theorem, since the use of those techniques requires a number of arithmetic and data manipulation steps.

Chapter 5 is divided into four sections. In the first section, we describe the theoretical basis of oblique elimination. We then use the theory to develop an algorithm, which we call minimal variable oblique elimination, for implementing oblique elimination in certain cases. In the second section, we develop necessary and sufficient conditions for the

minimal variable oblique elimination algorithm to be successful in computing tridiagonal decompositions of a given matrix. In the third section, we show that the minimum variable oblique elimination algorithm can be used to compute tridiagonal decompositions of the Fourier matrices. Finally, in the fourth section, we derive expressions for the parallel and serial operation counts of the algorithm implied by the decomposition. We also derive expressions for the number of operations required to compute the tridiagonal decompositions of a given matrix using the minimum variable oblique elimination algorithm.

5.1. Background and Description of Oblique Elimination

Recall that, as a special case of the results of Chapter 2, any square matrix over the real or complex numbers can be factored into a product of tridiagonal matrices. The proof of the theorem does not yield a good algorithm for doing so however, and, as mentioned in the Introduction, the problem of developing algorithms that will factor matrices in such a way is still unsolved in general. Tchuente has proposed a method called oblique elimination which will work in certain cases [61]. In this section we describe the theoretical basis developed by them and then show how an algorithm can be derived from the theory.

Throughout this chapter let n be an arbitrary positive integer with $n > 3$ and assume that all matrices and vectors are taken over the complex numbers. We could relax the last assumption but since we are mainly interested in the Fourier matrices we do not do so. In this chapter we consider matrices and vectors to be ordered from 1 to n rather than 0 to $n-1$ as in the last chapter. When referring to a matrix it will always be assumed that the entries of the matrix are denoted by the same letter as the matrix

unless explicitly stated otherwise.

Definition 5.1. Let A be an $n \times n$ lower triangular matrix. We say that A is *unit lower triangular* if $a_{ii} = 1$ for $i \in [1, n]$.

Definition 5.2. Let A be any $n \times n$ matrix. We say that A has an *LU decomposition* if there exists an $n \times n$ unit lower triangular matrix L and an upper triangular matrix U such that $A = LU$.

Computing the LU decomposition of a matrix will be the first step in the oblique elimination algorithm. Not every matrix has an LU decomposition. It can be shown that if A is any $n \times n$ matrix then there exists an $n \times n$ permutation matrix P such that PA has an LU decomposition [15, 61]. The most common method used for computing LU decompositions is Gaussian elimination with partial pivoting. A discussion of these topics can be found in almost any numerical analysis book [15, 59]. We remark that even if a matrix A has an LU decomposition it may be desirable, for reasons of numerical stability, to compute the LU decomposition of PA for some permutation matrix P rather than that of A .

Definition 5.3. Let A be an $n \times n$ lower triangular matrix. Let $i \in [1, n]$. The i^{th} *oblique* of A is the set $\{a_{n-i+1,1}, a_{n-i+2,2}, \dots, a_{n,i}\}$.

Note that we have defined the obliques relative to the lower triangular matrices. We could just as well have done so for upper triangular matrices. Throughout this chapter we will work mainly with lower triangular matrices. The techniques can all be applied to the transposes of the upper triangular matrices that appear in the decompositions.

Definition 5.4. Let A be an $n \times n$ matrix. We say that A has *lower bandwidth* r if $a_{ij} = 0$ whenever $i > j + r$.

An $n \times n$ lower triangular matrix A with lower bandwidth $n - j$ for some $j \in [1, n-1]$ has the property that the k^{th} oblique is $\{0\}$ for every $k \in [1, j]$.

The oblique elimination method as applied to an $n \times n$ matrix M can be summarized in the following two steps:

1.) Determine a permutation matrix P such that $PM = LU$ is an LU decomposition of PM .

2.) Construct matrices $L_1^{-1}, L_2^{-1}, \dots, L_{n-2}^{-1}$ such that for every $j \in [1, n-2]$ the matrix L_j is unit lower bidiagonal and the matrix $L_j^{-1}L_{j-1}^{-1} \cdots L_1^{-1}L$ has lower bandwidth $n - j$. Similarly, construct matrices $U_1^{-1}, U_2^{-1}, \dots, U_{n-2}^{-1}$ such that for every $j \in [1, n-2]$ the matrix U_j^t is unit lower bidiagonal and the matrix $(U_j^t)^{-1}(U_{j-1}^t)^{-1} \cdots (U_1^t)^{-1}U^t$ has lower bandwidth $n - j$. Then

$$A = L_{n-2}^{-1}L_{n-3}^{-1} \cdots L_1^{-1}L$$

$$B = UU_1^{-1}U_2^{-1} \cdots U_{n-2}^{-1}$$

are lower and upper bidiagonal matrices respectively. Thus,

$$M = P^{-1}L_1L_2 \cdots L_{n-2}ABU_{n-2}U_{n-3} \cdots U_1$$

is a tridiagonal decomposition of M . This methodology does not always work because one can not always construct the matrices U_j and L_j . A method for doing so is the following:

Let x_1, x_2, \dots, x_{n-1} denote indeterminates and let X denote the $n \times n$ matrix

$$X = \begin{bmatrix} 1 & 0 & . & . & . & . & 0 \\ -x_1 & 1 & 0 & . & . & . & . \\ 0 & -x_2 & 1 & . & . & . & . \\ . & 0 & x_3 & . & 0 & . & . \\ . & . & 0 & . & 1 & 0 & . \\ . & . & . & . & -x_{n-2} & 1 & 0 \\ 0 & . & . & . & 0 & -x_{n-1} & 1 \end{bmatrix}.$$

Then, since $I_n - X$ is nilpotent,

$$X^{-1} = (I - (I - X))^{-1} = \begin{bmatrix} 1 & 0 & . & . & . & 0 \\ x_1 & 1 & 0 & . & . & . \\ x_1 x_2 & x_2 & 1 & 0 & . & . \\ x_1 x_2 x_3 & x_2 x_3 & x_3 & 1 & . & . \\ . & . & . & . & . & . \\ . & . & . & . & 0 & . \\ . & . & . & . & 1 & 0 \\ x_1 x_2 \cdots x_{n-1} & x_2 \cdots x_{n-1} & x_3 \cdots x_{n-1} & . & x_{n-1} & 1 \end{bmatrix}.$$

Thus, one can attempt to construct the matrices $L_1^{-1}, L_2^{-1}, \dots, L_{n-2}^{-1}$ and $(U_j^t)^{-1}, (U_{j-1}^t)^{-1}, \dots, (U_1^t)^{-1}$ in the form of X^{-1} . Let A be an $n \times n$ lower triangular matrix with lower bandwidth $n-i+1$. Then there exists an $n \times n$ matrix X^{-1} as above such that the matrix $B = X^{-1}A$ has lower bandwidth $n-i$ if and only if the nonlinear system of equations

$$x_1 x_2 \cdots x_{n-i} a_{1,1} + x_2 x_3 \cdots x_{n-i} a_{2,1} + \cdots + x_{n-i} a_{n-i,1} + a_{n-i+1,1} = 0$$

$$x_2 x_3 \cdots x_{n-i+1} a_{2,2} + x_3 x_4 \cdots x_{n-i+1} a_{3,2} + \cdots + x_{n-i+1} a_{n-i+1,2} + a_{n-i+2,2} = 0$$

$$\vdots$$

$$x_i x_{i+1} \cdots x_{n-1} a_{i,i} + x_{i+1} x_{i+2} \cdots x_{n-1} a_{i+1,i} + \cdots + x_{n-1} a_{n-1,i} + a_{n,i} = 0$$

has a solution in the indeterminates x_1, x_2, \dots, x_{n-1} . This can be seen by writing the product $X^{-1}A$ out elementwise and setting the appropriate terms equal to zero.

In this dissertation we employ an algorithm which attempts to solve the system using a minimal number of variables. We call the algorithm *minimal variable oblique elimination*. If we take $x_1 = x_2 = \cdots = x_{n-i+1} = 0$, then the system reduces to

$$x_{n-i}a_{n-i,1} + a_{n-i+1,1} = 0$$

$$x_{n-i}x_{n-i+1}a_{n-i,2} + x_{n-i+1}a_{n-i+1,2} + a_{n-i+2,2} = 0$$

$$\cdot \quad \cdot \quad \cdot$$

$$x_{n-i}x_{n-i+1} \cdots x_{n-1}a_{n-i,i} + x_{n-i+1}x_{n-i+2} \cdots x_{n-1}a_{n-i+1,i} + \cdots + x_{n-1}a_{n-1,i} + a_{n,i} = 0.$$

A solution to this system, if it exists, is given by

$$x_{n-i} = -\frac{a_{n-i+1,1}}{a_{n-i,1}}$$

$$x_{n-i+1} = -\frac{a_{n-i+2,2}}{x_{n-i}a_{n-i,2} + a_{n-i+1,2}}$$

$$\cdot \quad \cdot \quad \cdot$$

$$x_{n-1} = -\frac{a_{n,i}}{x_{n-i}x_{n-i+1} \cdots x_{n-2}a_{n-i,i} + \cdots + x_{n-2}a_{n-2,i} + a_{n-1,i}}.$$

We shall refer to this particular solution set as S . Note that the existence of the solution set S is not equivalent to the existence of a solution to the original system of equations, even with $x_1 = x_2 = \cdots = x_{n-i+1} = 0$, since if A is the zero matrix, then the system is solved trivially but the solution set S is undefined. We shall concern ourselves with determining conditions under which this solution exists.

Definition 5.5. Let A be an $n \times n$ lower triangular matrix with lower bandwidth $n-i+1$.

If the above solution exists for A , then we call it the *minimal variable solution*. If X^{-1} is constructed from this solution in the fashion described in this section, then we say that $B = X^{-1}A$ is computed from A using minimal variable oblique elimination. X is called the *minimal variable solution matrix* for A .

Definition 5.6. Let L be an $n \times n$ lower triangular matrix and denote $L_0 \equiv L^{-1}$. We say that *minimal variable oblique elimination is successful for L* if there exists matrices L_1, L_2, \dots, L_{n-2} such that L_i is the minimal variable solution matrix for $L_{i-1}^{-1}L_{i-2}^{-1} \cdots L_1^{-1}L_0$ for every $i \in [1, n-2]$.

Definition 5.7. Let A be any $n \times n$ matrix. We say that *minimal variable oblique elimination is successful for A* if the following two conditions hold:

- 1.) A has an LU decomposition $A = LU$.
- 2.) Minimal variable oblique elimination is successful for L and U^t .

In the next section we derive necessary and sufficient conditions for minimal variable oblique elimination to be successful.

5.2. Necessary and Sufficient Conditions for the Minimal Variable Solution to Exist

In this section we derive necessary and sufficient conditions for the minimal variable solution to exist for a lower triangular banded matrix A . We will show that the solution exists if and only if certain submatrices of A are invertible. We then use these conditions to develop necessary and sufficient conditions for minimal variable oblique elimination to be successful for any square matrix.

We introduce some notation which will be used throughout this section. If B is a square matrix, then $\det(B)$ denotes the determinant of B . Fix $i \in [0, n]$ and let A be a lower triangular matrix with lower bandwidth $n-i+1$, that is,

$$A = \begin{bmatrix} a_{1,1} & 0 & . & . & . & . & . & 0 \\ . & a_{2,2} & . & . & . & . & . & . \\ a_{n-i+1,1} & . & . & . & 0 & . & . & . \\ 0 & a_{n-i+2,2} & . & . & a_{i,i} & . & . & . \\ . & 0 & . & . & . & . & . & . \\ . & . & . & . & . & . & . & 0 \\ 0 & 0 & . & 0 & a_{n,i} & . & . & a_{n,n} \end{bmatrix}.$$

For each $k \in [1, i]$, let M_k denote the $k \times k$ submatrix of A defined by

$$M_k \equiv \begin{bmatrix} a_{n-i,1} & a_{n-i,2} & . & . & a_{n-i,k-1} & a_{n-i,k} \\ a_{n-i+1,1} & a_{n-i+1,2} & . & . & a_{n-i+1,k-1} & a_{n-i+1,k} \\ 0 & a_{n-i+2,2} & . & . & a_{n-i+2,k-1} & a_{n-i+2,k} \\ 0 & 0 & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & 0 & . & 0 & a_{n-i+k-1,k-1} & a_{n-i+k-1,k} \end{bmatrix}$$

For convenience we take $M_0 \equiv (1)$. Note that the M_k are in upper Hessenberg form.

We will show that the minimum variable solution exists for A if and only if the M_k are all invertible.

For every $k \in [0, i-1]$ we formally define the symbol d_{n-i+k} by

$$d_{n-i+k} \equiv \left(\prod_{j=0}^{k-1} x_{n-i+j} \right) a_{n-i,k+1} + \left(\prod_{j=1}^{k-1} x_{n-i+j} \right) a_{n-i+1,k+1} + \cdots + x_{n-i+k-1} a_{n-i+k-1,k+1} + a_{n-i+k,k+1}.$$

The above definition is formal in the sense that if upon substituting values for the symbols x_{n-i+j} in a sequential fashion it happens that $d_{n-i+j} = 0$ for some j , then x_{n-i+k} is undefined for $k \geq j$. To show that the minimal variable solution exists it must be shown that the d_{n-i+k} are all defined and nonzero.

In what follows, some of the proofs are by induction. For purposes of illustrating the reasons for the truth of the theorems, the first several cases are sometimes shown to be true even though this is not a logical necessity.

Lemma 5.8. Assume that there exists an $l \in [1, i]$ such that $\det(M_k) \neq 0$ for every $k \in [0, l-1]$. Then for every such k , d_{n-i+k} exists and $d_{n-i+k} = \frac{\det(M_{k+1})}{\det(M_k)}$.

Proof.

The proof is by induction on k . If $k = 0$ then

$$d_{n-i} = a_{n-i,1} = \frac{\det(M_1)}{\det(M_0)}.$$

If $l = 1$ then we are done. Otherwise $d_{n-i} \neq 0$ by assumption. If $k = 1$ then

$$d_{n-i+1} = -\frac{a_{n-i+1,1}}{a_{n-i,1}}a_{n-i,2} + a_{n-i+1,2} = \frac{\det(M_2)}{\det(M_1)}.$$

Assume that the lemma is true for d_{n-i} , d_{n-i+1} , ..., $d_{n-i+k-1}$ for some $k \in [1, l-1]$.

Then each x_{n-i+j} for $j \in [0, k-1]$ is defined so d_{n-i+k} is defined and is given by

$$d_{n-i+k} = \left(\prod_{j=0}^{k-1} x_{n-i+j} \right) a_{n-i,k+1} + \left(\prod_{j=1}^{k-1} x_{n-i+j} \right) a_{n-i+1,k+1} + \cdots + x_{n-i+k-1} a_{n-i+k-1,k+1} + a_{n-i+k,k+1}.$$

By the induction hypothesis,

$$\begin{aligned} d_{n-i+k} &= (-1)^k \left[\frac{a_{n-i+1,1}}{\det(M_1)} \right] \left[\frac{a_{n-i+2,2}}{\det(M_2)} \right] \left[\frac{a_{n-i+3,3}}{\det(M_3)} \right] \cdots \left[\frac{a_{n-i+k,k}}{\det(M_k)} \right] a_{n-i,k+1} + \\ &+ (-1)^{k-1} \left[\frac{a_{n-i+2,2}}{\det(M_2)} \right] \left[\frac{a_{n-i+3,3}}{\det(M_3)} \right] \cdots \left[\frac{a_{n-i+k,k}}{\det(M_k)} \right] a_{n-i+1,k+1} + \\ &+ \cdots + (-1) \left[\frac{a_{n-i+k,k}}{\det(M_k)} \right] a_{n-i+k-1,k+1} + a_{n-i+k,k+1} = \\ &= \frac{(-1)^k \left[\prod_{j=1}^k a_{n-i+j,j} \right] a_{n-i,k+1} + (-1)^{k-1} \left[\prod_{j=2}^k a_{n-i+j,j} \right] a_{n-i+1,k+1} + \cdots}{\det(M_k)} \end{aligned}$$

$$\frac{\dots + (-1)\det(M_{k-1})a_{n-i+k,k}a_{n-i+k-1,k+1} + \det(M_k)a_{n-i+k,k+1}}{\det(M_k)}$$

Since M_{k+1} is in upper Hessenberg form, the numerator of the last expression is $\det(M_{k+1})$ expanded along the column $k+1$. This proves the lemma.

Q.E.D.

Lemma 5.9. *Assume that there exists an $l \in [0, i-1]$ such that the d_{n-i+k} are defined and nonzero for every $k \in [0, l]$. Then $\det(M_k) \neq 0$ for every $k \in [0, l]$.*

Proof.

The proof is by induction on k . If $k = 0$, then $\det(M_0) = 1$. If $l = 0$, then we are done. Otherwise if $k = 1$, then $\det(M_1) = a_{n-i,1} = d_{n-i} \neq 0$. If $k = 2$ then

$$d_{n-i+1} = -\frac{a_{n-i+1,1}}{a_{n-i,1}}a_{n-i,2} + a_{n-i+1,2} = -\frac{1}{a_{n-i,1}}\det(M_2) \neq 0.$$

Assume that the lemma is true for M_0, M_1, \dots, M_{k-1} for some $k \in [2, l]$. Since $\det(M_0), \det(M_1), \dots, \det(M_{k-1}) \neq 0$, by the calculations of the previous lemma,

$$d_{n-i+k-1} = \frac{\det(M_k)}{\det(M_{k-1})}.$$

By hypothesis, $d_{n-i+k-1} \neq 0$ so we may conclude that $\det(M_k) \neq 0$ which proves the lemma.

Q.E.D.

Lemmas 5.8 and 5.9 can be combined to yield necessary and sufficient conditions for the minimal variable solution to exist.

Theorem 5.10. The minimal variable solution exists for A if and only if the matrices M_1, M_2, \dots, M_i are invertible. In this case we have

$$x_{n-i+k} = -a_{n-i+k+1,k+1} \left[\frac{\det(M_k)}{\det(M_{k+1})} \right]$$

for every $k \in [0, i-1]$.

Proof.

Assume that the minimal variable solution exists for A. Then $d_{n-i+k} \neq 0$ for every $k \in [0, i-1]$. By Lemma 5.9, $\det(M_k) \neq 0$ for every $k \in [0, i-1]$. By Lemma 5.8, $d_{n-i+k} = \frac{\det(M_{k+1})}{\det(M_k)}$ which yields the desired expression for x_{n-i+k} .

Conversely, assume that the matrices M_1, M_2, \dots, M_i are invertible. Then, by Lemma 5.8, for every $k \in [0, i-1]$, d_{n-i+k} exists and is nonzero. Hence, the minimal variable solution exists.

Q.E.D.

We now assume that $L \equiv L_0$ is lower triangular and that A is of the form $A = L_{i-1}^{-1} L_{i-2}^{-1} \cdots L_1 L_0$ where L_1, L_2, \dots , and L_{i-1} have been constructed using minimal variable oblique elimination. In the next theorem, we establish criteria which will allow us to deduce the existence (or non-existence) of the minimal variable solution for A using information contained in the matrix L. We will show that, due to the sparse structure of the L_i , one of the matrices M_k will be singular if and only if certain submatrices of L are singular.

Definition 5.11. If L is an $n \times n$ lower triangular matrix, then for every pair of integers (j, k) with $j \in [1, n-1]$ and $k \in [1, n-j]$ define

$$L_{kj} = \begin{bmatrix} l_{j,1} & \cdot & \cdot & l_{j,k} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ l_{j+k-1,1} & \cdot & \cdot & l_{j+k-1,k} \end{bmatrix}.$$

Similarly, if U is an $n \times n$ upper triangular matrix, then define

$$U_{kj} \equiv [(U^t)_{kj}]^t.$$

Theorem 5.12. Assume that $L = L_0$ is an $n \times n$ lower triangular matrix and that either $i = 1$ or $i \in [2, n-2]$ and $n \times n$ matrices L_1, L_2, \dots , and L_{i-1} have been constructed using minimal variable oblique elimination. Suppose that $A = L_{i-1}^{-1} L_{i-2}^{-1} \cdots L_1^{-1} L_0$ and that the minimal variable solution does not exist for A . Let j be the smallest positive integer such that M_j is singular. Then $L_{j,n-i}$ is singular.

Proof.

We show that $\det(L_{j,n-i}) = 0$. By construction, for $k \in [1, i-1]$,

$$L_k = \left[\begin{array}{c|cccccc} I_{n-k} & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ \hline 0 \cdot \cdot \cdot 0 & -x_{n-k}^{(k)} & 1 & 0 & \cdot & \cdot & 0 \\ \hline 0 & -x_{n-k+1}^{(k)} & 1 & 0 & \cdot & \cdot & 0 \\ 0 & 0 & -x_{n-k+2}^{(k)} & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 0 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & \cdot & \cdot & -x_{n-1}^{(k)} & 1 \end{array} \right].$$

where $\{-x_{n-k+j}^{(k)}\}_{j=0}^{k-1}$ denotes the minimal variable solutions for the matrix $L_{k-1}^{-1} L_{k-2}^{-1} \cdots L_1^{-1} L$. We use the notation $R_k \rightarrow R_k + \alpha R_m$ to express the fact that row k of a matrix is replaced by itself plus a multiple, α , of row m . If B is any $n \times n$ matrix and $k \in [1, i-1]$ then multiplying B on the left by L_k has the effect:

$$\begin{aligned} R_m &\rightarrow R_m && \text{for } m \in [1, n-k] \\ R_m &\rightarrow R_m - x_{m-1}^{(k)} R_{m-1} && \text{for } m \in [n-k+1, n] \end{aligned}$$

Thus, multiplying A on the left by L_{i-1} leaves rows 1 to $n-i+1$ fixed and replaces rows $n-i+2$ to n of A with a linear combination of them and the row directly above them, multiplying $L_{i-1}A$ on the left by L_{i-2} leaves rows 1 to $n-i+2$ fixed and replaces rows $n-i+3$ to n of $L_{i-1}A$ with a linear combination of them and the row directly above them, ..., multiplying $L_{i-j+1} \cdots L_{i-1}A$ on the left by L_{i-j} leaves rows 1 to $n-i+j$ fixed and replaces rows $n-i+j$ to n of $L_{i-j+1} \cdots L_{i-1}A$ with a linear combination of them and the row directly above them. Since the other left multiplications by L_1, L_2, \dots, L_{i-j} leave rows $n-i, n-i+1, \dots, n-i+j-1$ of $L_{i-j+1} \cdots L_{i-1}A$ unchanged and $L = \left[\prod_{m=1}^{i-1} L_m \right] A$ it follows that rows $n-i, n-i+1, \dots, n-i+j-1$ of L are linear combinations of rows $n-i, n-i+1, \dots, n-i+j-1$ of A . In fact, we can write $L_{j,n-i} = \hat{L}_1 \hat{L}_2 \cdots \hat{L}_{i-1} M_j$ where

$$\hat{L}_k = \left[\begin{array}{c|cccccc} \hat{L}_{i-k} & 0 & 0 & . & . & . & 0 \\ \hline 0 & . & 0 & -x_{n-k}^{(k)} & & & \\ \hline 0 & & 0 & -x_{n-k+1}^{(k)} & 1 & 0 & . & . & 0 \\ 0 & & 0 & & -x_{n-k+2}^{(k)} & 1 & . & . & . \\ . & & . & & 0 & . & . & . & . \\ . & & . & & . & . & . & . & 0 \\ 0 & & 0 & & 0 & . & . & -x_{k+j-i}^{(k)} & 1 \end{array} \right]$$

That is, \hat{L}_k is the submatrix of L_k occupying the same position as M_j does in A . Since each \hat{L}_k is unit lower triangular, $\det(\hat{L}_k) = 1$. Hence, $\det(L_{j,n-i}) = \det(M_j) = 0$ which proves the theorem.

Q.E.D.

Note that if $M = LU$ is an LU decomposition of a $n \times n$ matrix M , then the theorem holds for both L and U^t .

We are now in a position to state necessary and sufficient conditions for minimal variable oblique elimination to be successful for an arbitrary $n \times n$ matrix. It is immediate from the definition that the matrix must have an LU decomposition. Therefore, we assume this condition in the next theorem.

Theorem 5.13. *Assume that B is an $n \times n$ matrix with LU decomposition $B = LU$. Minimal variable oblique elimination is successful for B if and only if the submatrices L_{kj} and U_{kj} of L and U are invertible for every $j \in [2, n-1]$ and $k \in [1, n-j]$.*

Proof.

Assume that the minimal variable oblique elimination method is successful for B and assume by way of contradiction that there exists $j \in [1, n-1]$ and $k \in [1, n-j]$ such that L_{kj} is not invertible. Denote $L \equiv L_0$. By Definitions 5.6 and 5.7, there exists matrices L_1, L_2, \dots, L_{n-2} such that for $i \in [1, n-2]$, L_i is the minimal variable solution matrix for $L_{i-1}^{-1}L_{i-2}^{-1} \cdots L_1^{-1}L$. Let $A = L_{n-j-1}^{-1}L_{n-j-2}^{-1} \cdots L_1^{-1}L$. As was shown in the proof of Theorem 5.12, $\det(L_{kj}) = \det(M_k)$. By Theorem 5.10, M_k is invertible so $\det(M_k) \neq 0$. Since L_{kj} is assumed to be singular, $\det(L_{kj}) = 0$ which is a contradiction. The same argument can be applied to U_{kj} .

Conversely, assume that for every $j \in [1, n-1]$ and $k \in [1, n-j]$ the matrices L_{kj} and U_{kj} are invertible and that minimal variable oblique elimination is not successful for B . Then it is not successful for either L or U^t . Assume that it is not successful for L . Let i be such that we are able to construct the matrices L_1, L_2, \dots, L_{i-1} using minimal variable oblique elimination and such that the minimal variable solution does not exist for $A =$

$L_{j-1}^{-1}L_{j-2}^{i-2} \cdots L_1^{-1}L$. Let j be the smallest integer such that M_j is not invertible. By Theorem 5.12, $L_{j,n-j}$ is not invertible. By definition of M_j , $1 \leq j \leq i \leq n-1$ so $n-j \geq 1$. Therefore, $L_{j,n-j}$ is invertible which is a contradiction. If minimal variable oblique elimination is not successful for U^t , then the same argument can be applied.

Q.E.D.

We have developed criteria which can be applied to the factors of the LU decomposition of a matrix to determine whether or not oblique elimination will be successful for that matrix. This criteria is much easier to use than checking to see if the original system of nonlinear equations has a solution at each step. The conditions stated in Theorem 5.13 are stringent. One can easily think of a great many examples of matrices which do not satisfy the conditions. Fortunately, all of the Fourier matrices do satisfy the conditions. In the next section we prove this assertion.

5.3. Application of Minimal Variable Oblique Elimination to the Fourier Matrices

In this section we show that minimal variable oblique elimination is successful for any Fourier matrix. In fact, we show that the technique is successful for PF_n where P is any permutation matrix. We first show that PF_n has an LU decomposition for any permutation matrix P . We then show that these LU decompositions all satisfy the conditions of Theorem 5.13 of the previous section. Recall that we denote $\omega \equiv \omega_n \equiv \exp[-2\pi i/n]$ where $i = \sqrt{-1}$.

Definition 5.14. Let B be any $n \times n$ matrix and let $k \in [1, n]$. The *leading $k \times k$ principal submatrix* of B is the matrix

$$B_k \equiv \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,k} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{k,1} & b_{k,2} & \cdots & b_{k,k} \end{bmatrix}.$$

Lemma 5.15. Let $k \in [1, n]$ and let $s_1, s_2, \dots, s_k \in [0, n-1]$ be distinct. Let $j \in [0, n-1]$ such that $j+k-1 < n$. For each $i \in [1, k]$, define

$$\vec{v}_i \equiv (\omega^{(j+i-1)s_1}, \omega^{(j+i-1)s_2}, \dots, \omega^{(j+i-1)s_k}).$$

The set $\{\vec{v}_i\}_{i=1}^k$ is a linearly independent set.

Proof.

Assume that the lemma is false, that is, that there exists constants c_1, c_2, \dots, c_k not all zero such that $\sum_{i=1}^k c_i \vec{v}_i = 0$. Writing the sum out yields

$$\omega^{js_1} (c_1 + c_2 \omega^{s_1} + c_3 \omega^{2s_1} + \cdots + c_k \omega^{(k-1)s_1}) = 0$$

$$\omega^{js_2} (c_1 + c_2 \omega^{s_2} + c_3 \omega^{2s_2} + \cdots + c_k \omega^{(k-1)s_2}) = 0$$

$$\vdots$$

$$\omega^{js_k} (c_1 + c_2 \omega^{s_k} + c_3 \omega^{2s_k} + \cdots + c_k \omega^{(k-1)s_k}) = 0.$$

Define a polynomial $p(x) \in \mathbb{C}[x]$ by $p(x) = c_1 + c_2 x + \cdots + c_k x^{k-1}$. Then, since $s_i \neq s_j$ if $i \neq j$ and $s_i \in [0, n-1]$ for every $i \in [1, k]$, the set $\{\omega^{s_1}, \omega^{s_2}, \dots, \omega^{s_k}\}$ is a set of k distinct roots of $p(x)$. By definition of $p(x)$, $\deg(p(x)) \leq k-1$ which implies that $p(x)$ is the zero polynomial. Therefore, $c_1 = c_2 = \cdots = c_k = 0$ which is a contradiction.

Q.E.D.

Theorem 5.16. Let P be an $n \times n$ permutation matrix. Every leading $k \times k$ principal sub-

matrix of PF_n is invertible.

Proof.

Assume that P represents the permutation σ , that is, $P = P_\sigma$. Since in this chapter we consider matrices and vectors to be ordered from 1 to n rather than 0 to $n-1$, we consider σ as acting on the set $\{1, 2, \dots, n\}$. Denote

$$F_n = \begin{bmatrix} \vec{f}_1 \\ \vec{f}_2 \\ \vdots \\ \vec{f}_n \end{bmatrix}.$$

where $\vec{f}_i \equiv (1, \omega^{i-1}, \omega^{2(i-1)}, \dots, \omega^{(n-1)(i-1)})$. Then

$$PF_n = \begin{bmatrix} \vec{f}_{\sigma(1)} \\ \vec{f}_{\sigma(2)} \\ \vdots \\ \vec{f}_{\sigma(n)} \end{bmatrix}.$$

Let $k \in [1, n]$ and let B_k denote the leading $k \times k$ principal submatrix of PF_n . Thus, letting $s_i \equiv \sigma(i)-1$,

$$B_k = \begin{bmatrix} 1 & \omega^{s_1} & \omega^{2s_1} & \dots & \omega^{(k-1)s_1} \\ 1 & \omega^{s_2} & \omega^{2s_2} & \dots & \omega^{(k-1)s_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{s_k} & \omega^{2s_k} & \dots & \omega^{(k-1)s_k} \end{bmatrix}.$$

Since σ is a permutation, the s_i are distinct. By Lemma 5.15, the columns of B_k are linearly independent.

Q.E.D.

Corollary 5.17. Let P be an $n \times n$ permutation matrix. The matrix PF_n has an LU decomposition. In particular, F_n has an LU decomposition.

Proof.

Theorem 5.16 shows that PF_n satisfies the required conditions [15].

Q.E.D.

Theorem 5.18. Let P be an $n \times n$ permutation matrix and assume that $PF_n = LU$ is the LU decomposition of PF_n . Let $j \in [1, n-1]$, $k \in [1, n-j]$, and L_{jk} and U_{jk} be as in Definition 5.11. The matrices L_{jk} and U_{jk} are invertible.

Proof.

We first show that L_{jk} is invertible. Denote

$$\vec{u}_{1k} \equiv (u_{11}, 0, \dots, 0)^t$$

$$\vec{u}_{2k} \equiv (u_{12}, u_{22}, 0, \dots, 0)^t$$

$$\vec{u}_{3k} \equiv (u_{13}, u_{23}, u_{33}, 0, \dots, 0)^t$$

$$\vdots$$

$$\vec{u}_{kk} \equiv (u_{1k}, u_{2k}, u_{3k}, \dots, u_{kk})^t.$$

Since F_n is nonsingular and L is unit lower triangular, $\det(F_n) = \det(P^{-1})\det(L)\det(U) = \det(P^{-1})\det(U) \neq 0$. Hence, $\det(U) = \prod_{i=1}^n u_{ii} \neq 0$. This implies that the scalars $u_{11}, u_{22}, \dots, u_{kk}$ are nonzero and therefore that $\{\vec{u}_{ik}\}_{i=1}^k$ is a linearly independent set. Since a square matrix is invertible if and only if it takes a basis to a basis, it is sufficient to show that $\{L_{jk}\vec{u}_{ik}\}_{i=1}^k$ is a linearly independent set.

For $i \in [1, k]$, let $\bar{v}_i \equiv (v_{1i}, v_{2i}, \dots, v_{ki})^t \equiv L_{jk} \bar{u}_{ik}$. Since L_{jk} is $k \times k$ and the first k columns of U have zeroes in rows $k+1, k+2, \dots, n$, v_{mi} is the $(j+m-1, i)$ element of PF_n . Hence, letting $s_m \equiv \sigma(j+m-1)-1$ so that $v_{mi} = \omega^{(i-1)s_m}$, we have

$$\begin{aligned}\bar{v}_1 &= (1, 1, \dots, 1)^t \\ \bar{v}_2 &= (\omega^{s_1}, \omega^{s_2}, \dots, \omega^{s_k})^t \\ \bar{v}_3 &= (\omega^{2s_1}, \omega^{2s_2}, \dots, \omega^{2s_k})^t \\ &\vdots \\ \bar{v}_k &= (\omega^{(k-1)s_1}, \omega^{(k-1)s_2}, \dots, \omega^{(k-1)s_k})^t\end{aligned}$$

By Lemma 5.15 with $j = 0$, $\{\bar{v}_i\}_{i=1}^k$ is a linearly independent set. Hence, L_{jk} is invertible.

We now show that U_{jk} is invertible. The argument is almost the same but some modifications are necessary. Denote

$$\begin{aligned}\bar{l}_{1k} &\equiv (l_{11}, 0, \dots, 0)^t \\ \bar{l}_{2k} &\equiv (l_{12}, l_{22}, 0, \dots, 0)^t \\ \bar{l}_{3k} &\equiv (l_{13}, l_{23}, l_{33}, 0, \dots, 0)^t \\ &\vdots \\ \bar{l}_{kk} &\equiv (l_{1k}, l_{2k}, l_{3k}, \dots, l_{kk})^t.\end{aligned}$$

Since L is unit lower triangular, $\{\bar{l}_{ik}\}_{i=1}^k$ is a linearly independent set. We show that $\{U_{jk} \bar{l}_{ik}\}_{i=1}^k$ is a linearly independent set. For $i \in [1, k]$, let $\bar{v}_i \equiv (v_{1i}, v_{2i}, \dots, v_{ki})^t \equiv U_{jk} \bar{l}_{ik}$. Then v_{mi} is the $(j+m-1, i)$ element of $(PF_n)^t$ and is therefore the $(i, j+m-1)$ element of PF_n . Hence, letting $s_i \equiv \sigma(i)-1$, we have

$$\bar{v}_1 = (\omega^{js_1}, \omega^{(j+1)s_1}, \dots, \omega^{(j+k-1)s_1})^t$$

$$\begin{aligned}\vec{v}_2 &= (\omega^{js_2}, \omega^{(j+1)s_2}, \dots, \omega^{(j+k-1)s_2})t \\ &\vdots \\ \vec{v}_k &= (\omega^{js_k}, \omega^{(j+1)s_k}, \dots, \omega^{(j+k-1)s_k})t\end{aligned}$$

Let M be the $k \times k$ matrix having \vec{v}_i in the i^{th} row. Since the column rank of a matrix is equal to the row rank, the \vec{v}_i will be linearly independent if and only if the columns of M are. By Lemma 5.15, the columns of M are linearly independent. Hence, U_{jk} is invertible.

Q.E.D.

The next corollary is the main result of this section.

Corollary 5.19. *Let P be an $n \times n$ permutation matrix. Minimal variable oblique elimination is successful for PF_n ; that is, there exists unit lower bidiagonal matrices L_1, L_2, \dots, L_{n-2} , unit upper bidiagonal matrices U_1, U_2, \dots, U_{n-2} , and bidiagonal matrices B and C such that*

$$PF_n = L_1 L_2 \cdots L_{n-2} B C U_{n-2} U_{n-3} \cdots U_1.$$

Proof.

By Theorem 5.18, for every $j \in [1, n]$ and $k \in [1, n-j+1]$, the matrices L_{jk} and U_{jk} are invertible. By Theorem 5.13, this implies that minimal variable oblique elimination is successful for PF_n .

Q.E.D.

The next theorem is of interest for implementation purposes.

Theorem 5.20. *For every $n > 2$, the matrices B and C in Theorem 5.19 can be chosen*

such that $U_j = L_j^t$ for $j \in [1, n-2]$, that is,

$$F_n = L_1 L_2 \cdots L_{n-2} B C L_{n-2}^t L_{n-3}^t \cdots L_1^t.$$

Proof.

Let $F_n = LU$ be the LU decomposition of F_n . Since F_n is symmetric, $U = D L^t$ where D is a diagonal matrix [15]. Suppose that $L = L_1 L_2 \cdots L_{n-2} B$ is the tridiagonal decomposition of L computed using minimal variable oblique elimination. Then $U^t = L D = L_1 L_2 \cdots L_{n-2} B D$. Take $C = D B^t$.

Q.E.D.

We have shown that minimal variable oblique elimination can be used to compute tridiagonal decompositions of the Fourier matrices. The decompositions need to be computed only once for each n . The factors of the decomposition can then be stored permanently. Thus, a library of parallel algorithms for computing discrete Fourier transforms of any size (within some upper bound) should be easy to construct using this technique. The same program can be used for any n , which is far from the case with FFT-based methods.

5.4. Complexity Considerations

In this section we derive expressions for the time complexity of the parallel and serial algorithms resulting from the decompositions developed in this chapter. We also establish upper bounds on the number of floating point operations required to compute the decompositions using minimal variable oblique elimination. Recall that the algorithms resulting from the decompositions are generally required to be real-time algorithms whereas the computation of the tridiagonal factors need not be.

Theorem 5.21. *Let M be an $n \times n$ matrix and assume that minimal variable oblique elimination is successful for M . It takes $2n-2$ parallel steps, all steps but one consisting of, at most, one multiplication and one addition per point and the other step consisting of at most two multiplications and one addition per point, to implement the transformation $\vec{v} \rightarrow M\vec{v}$ using the tridiagonal decomposition of M .*

Proof.

The result follows immediately from Corollary 5.19.

Q.E.D.

We shall use the concept of a *flop* (floating point operation) to quantify the complexity of a serial computation. Golub and van Loan define a flop to be “more or less the work associated with the statement $s := s + a_{ik}b_{kj}$ ” [15]. We take s , a_{ik} and b_{kj} to be complex numbers. Note that it takes at least three real multiplications and five additions or four real multiplications and two additions to perform a complex multiplication [6].

Theorem 5.22. *Let M be an $n \times n$ matrix and assume that minimal variable oblique elimination is successful for M . The mapping $\vec{v} \rightarrow M\vec{v}$ can be accomplished with n^2+2 flops using the decomposition resulting from oblique elimination.*

Proof.

If $\vec{x} \in \mathbb{C}^n$ and $\vec{y} = L_i\vec{x}$ for some $i \in [1, n-2]$, then, by the way the L_i are constructed, it takes one flop to compute y_k if $k \geq n-i+1$, and zero otherwise. The same statement holds true for the U_i . If $\vec{y} = C\vec{x}$, then it takes two flops to compute y_k for $k \in [1, n]$. If $\vec{y} = B\vec{x}$, then, since B is unit lower bidiagonal, it takes one flop to compute y_k for $k \in$

$[1, n]$. Hence, it takes a total of $2 \sum_{i=1}^{n-2} i + 3n = n^2 + 2$ flops.

Q.E.D.

The operation count associated with using the tridiagonal decompositions to implement linear transforms on a serial machine is essentially the same as the operation count of the straightforward method. Therefore, these decompositions do not lead to good serial algorithms for computing linear transforms in general.

We now show that if minimal variable oblique elimination is successful, then it takes $O(n^3)$ flops to compute the decomposition. The computations required to compute the decompositions can be divided into three categories. One category consists of the computations necessary to compute the LU decomposition. The other categories consist of the computations necessary to construct the minimal variable solution matrices and the matrix multiplications necessary to compute the intermediate results, that is, the multiplications of the form $L_i^{-1}(L_{i-1}^{-1} \cdots L_1^{-1}L)$. The first category is well studied. We examine the latter two categories.

We first consider the computations necessary to construct the minimal variable solution matrices. In the next theorem, we establish an upper bound on the number of flops required to compute the minimal variable solution matrix for a lower triangular, banded matrix A assuming that a Horner type algorithm is used.

Theorem 5.23. *Let $i \in [1, n-2]$ and let A be an $n \times n$ matrix with lower bandwidth $n-i+1$. For each $k \in [0, i-1]$ assume that x_{n-i+k} is computed by first computing d_{n-i+k} in the nested fashion*

$$d_{n-i+k} = x_{n-i+k-1} (x_{n-i+k-2} (\cdots (x_{n-i+1} (x_{n-i} a_{n-i, k+1} + a_{n-i+1, k+1}) +$$

$$+ a_{n-i+2,k+1}) + \cdots + a_{n-i+k-1,k+1}) + a_{n-i+k,k+1} a_{n-i+k,k+1}$$

and then computing

$$x_{n-i+k} = - \frac{a_{n-i+k+1,k+1}}{d_{n-i+k}}.$$

It takes $\frac{i(i+1)}{2}$ flops to compute the minimal variable solution for A.

Proof.

It takes k flops to compute d_{n-i+k} , one for each x_{n-i+j} , $j = 0, 1, \dots, k-1$. It takes one more division to compute x_{n-i+k} . Hence, it takes $\sum_{k=0}^{i-1} (k+1) = \frac{i(i+1)}{2}$ flops altogether.

Q.E.D.

Corollary 5.24. Let M be an $n \times n$ matrix and assume that minimal variable oblique elimination is successful for M. Using the method of Theorem 5.23, it takes $\frac{1}{3}n(n-1)(n-2) = o(n^3)$ flops to construct the minimal variable solution matrices required to factor M.

Proof.

The solution matrices must be computed for $A_1 = L_1^{-1}L$, $A_2 = L_2^{-1}L_1^{-1}L$, \dots , $A_{n-2} = L_{n-2}^{-1} \cdots L_1^{-1}L$. It takes $\frac{i(i+1)}{2}$ flops to do so for each A_i . The same must be done for U^t . Hence, it takes $2 \sum_{i=1}^{n-2} \frac{i(i+1)}{2} = \sum_{i=1}^{n-2} i^2 + \sum_{i=1}^{n-2} i = \frac{1}{3}n(n-1)(n-2)$ flops altogether.

Q.E.D.

We now derive upper bounds on the number of flops required to carry out the matrix multiplications needed to construct the intermediate results. Recall that if $i \in [1, n-2]$, then

$$L_i^{-1} = \left[\begin{array}{c|cccccc} I_{n-i-1} & & & & & 0 \\ \hline & 1 & 0 & . & . & . & 0 \\ & x_{n-i} & 1 & 0 & . & . & . \\ 0 & x_{n-i}x_{n-i+1} & x_{n-i+1} & 1 & . & . & . \\ & . & . & . & . & 1 & . \\ & \prod_{j=0}^{i-1} x_{n-i+j} & \prod_{j=1}^{i-1} x_{n-i+j} & . & . & x_{n-1} & 1 \end{array} \right].$$

Theorem 5.25. Let $i \in [1, n-2]$ and let $A = L_i^{-1} \cdots L_1^{-1}L$. Let R_k denote the k^{th} row of A .

Assume that the matrix multiplication is computed in the following nested fashion:

$$R_{n-i+j} \rightarrow x_{n-i+j-1} (x_{n-i+j-2} (\cdots x_{n-i+1} (x_{n-i}R_{n-i} + R_{n-i+1}) + \\ + R_{n-i+2}) + \cdots + R_{n-i+j-1}) + R_{n-i+j}.$$

for $j \in [1, i]$. Then the matrix multiplication can be carried out using $i(n-i+2)$ flops.

Proof.

The multiplication leaves rows 1 to $n-i+1$ unchanged. The computation of the $n-i+j^{\text{th}}$ row is of the form $R_{n-i+j} \rightarrow x_{n-i+j-1}E + R_{n-i+j}$ where E is the value used to compute row $n-i+j-1$. Therefore, it takes one flop per row element to compute the new row. Since A is lower triangular with lower bandwidth $n-i+1$, there are at most $n-i+1$ nonzero elements in each row. Furthermore, the nonzero part of each row is offset by one location from the nonzero part of the row directly above and below it. Hence, the linear combination needs to be carried out for only $n-i+2$ elements in each row. Since only i rows are changed, the matrix multiplication can be carried out in $i(n-i+2)$ flops.

Q.E.D.

Corollary 5.26. *Let M be an $n \times n$ matrix and assume that minimal variable oblique elimination is successful for M . Let $M = LU$ be the LU decomposition of M . The matrix multiplications necessary to compute the minimal variable tridiagonal decompositions of L and U can be carried out using $\frac{1}{3}(n-1)(n-2)(n+9)$ flops.*

Proof.

The multiplications $L_1^{-1}L$, $L_2^{-1}(L_1^{-1}L)$, \dots , and $L_{n-2}^{-1}(L_{n-3}^{-1} \cdots L_1^{-1}L)$, each of which can be done using $i(n-i+2)$ flops, must be computed. The same must be done for U^t .

Hence it takes a total of $2 \sum_{i=1}^{n-2} i(n-i+2) = \frac{1}{3}(n-1)(n-2)(n+9)$ flops.

Q.E.D.

We now combine the results obtained in this chapter to point out that oblique elimination is, at most, an $O(n^3)$ operation.

Corollary 5.27. *If minimal variable oblique elimination is successful for the $n \times n$ matrix M , then it can be carried out using $O(n^3)$ flops.*

Proof.

By Corollaries 5.24 and 5.26, it takes, at most, $O(n^3)$ flops to compute the tridiagonal decompositions of L and U . Standard algorithms can be used to compute the LU decomposition in $O(n^3)$ flops. Thus, the whole procedure takes $O(n^3)$ flops.

Q.E.D.

In fact, each of the separate operation counts are of the form $\frac{1}{3}n^3 + O(n^2)$ so the total operation count is of the form $n^3 + O(n^2)$.

We have described minimal variable oblique elimination and shown that it can be used to compute tridiagonal decompositions of the Fourier matrices. The number of parallel steps required to compute a one-dimensional DFT on a linearly connected array of processors is $2n-2$ arithmetic steps and at most n permutation steps for every n . Moreover, since the permutation steps all occur at once and are at the end of the computation, it is possible that they can be done some other way.

CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

We have demonstrated that image algebra is useful as a model of parallel image processing and as a tool for the development of parallel algorithms for computing linear image to image transforms. We have shown that the algebraic relationships resulting from the establishment of an algebraic structure for digital image processing can yield useful results and techniques.

More specifically, we have shown that

1.) The image algebra provides an alternative algebraic formulation of linear image processing that reflects contemporary computing environments.

2.) Local decompositions of linear transforms exist in all "reasonable" cases; that is, we have shown that, given a network of processors interconnected by some network of communications links, any linear transformation can be factored into a sequence of linear transformations having the property that each transformation in the sequence is local with respect to the network if and only if there is a path between every pair of processors in the network.

3.) The image algebra can serve as an algebraic model for linear computations on computer networks based on group structures, Cayley networks, and provides a link between these networks and existing algebraic structures. We have characterized those linear transformations which are translation invariant with respect to a Cayley network and shown that the set of all such transformations is an algebra which is isomorphic to an existing algebraic object called the group algebra.

4.) Research in developing computer programs for factoring multivariable polynomials has applications to parallel image processing. In particular, convolutions can be decomposed into smaller convolutions by factoring polynomials corresponding to the images to be convolved.

5.) It is feasible to implement discrete Fourier transforms of sizes other than powers of two on mesh-connected arrays. We have developed algorithms and written programs to show how this can be done. We have provided estimates on the number of steps required to implement such algorithms on mesh-connected arrays.

6.) Numerical methods can be used to develop alternative methods for computing small discrete Fourier transforms locally, particularly on processors that are designed to implement floating point operations quickly. We have shown how this can be done and have provided FORTRAN programs which apply these numerical methods to the Fourier matrices to compute the coefficients required to implement DFTs locally using this technique and which use these coefficients to compute DFTs locally for arbitrary values of n less than a certain upper bound. These algorithms take $2n-2$ parallel steps to compute the DFT of length n for any n less than this upper bound and, at most, n elementary parallel permutation steps.

The existence theorem in Chapter 2 (Theorem 2.16) and the use of relationships between image algebra and other algebraic structures in developing local decompositions suggest that it may be possible, at some point in the near future, to build an image algebra compiler that is capable of decomposing classes of templates into products of local templates. The existence of such a compiler would ease the burden of developing parallel algorithms for programmers. On a smaller scale, the results presented in this dissertation would seem to imply that researchers should have the ability to develop computer

programs that would use algebraic methods to decompose computations into forms compatible with parallel architectures in the not too distant future. More research on general methods would need to be done for these things to become a reality.

We now list some suggestions for further research. Many of these suggestions are questions about which the author is curious. Therefore, if results are obtained that are related to the following problems, it would be appreciated if the author were made aware of them. We have attempted to present the problems in the same order as the material in the main body of the dissertation on which they are based. There is some overlap however.

One area of research is the generalization of the results in this dissertation to the \boxtimes and \circledast operations. Can an analogue of the existence theorem of Chapter 2 be formulated and proved? Miller [31] represents the Minkowski operations by convolution operations. The operations \boxtimes and \circledast are both related to the Minkowski operations. Can analogues of circulant templates be defined for \boxtimes and/or \circledast using Miller's work, or group algebras, as a guide? If so, is there an analogue of the discrete Fourier transform of the FFT? Given affirmative answers to these questions, are there methods analogous to those given in Chapter 4 for deriving local decompositions of templates with respect to \boxtimes and \circledast ?

It has been shown that the algebra generated by inversion and composition of Toeplitz integral operators is dense in the space of arbitrary kernels and, therefore, that if $K(t,u)$ is an arbitrary $p \times p$ kernel matrix, then the mapping $f \rightarrow g$ defined by

$$g(t) \equiv \int_0^{\pi} K(t,u)f(u)du \quad 0 \leq t \leq T.$$

can be approximated using FFTs or fast convolution algorithms [23]. Interpret this result in the setting of image algebra. Use it and any related results to develop efficient methods for approximating arbitrary linear image to image transforms by fast, local transforms. Investigate methods applicable to special classes of linear transforms. Recall that Schwartz has deemed this a problem of great interest in robot vision [50]. Find out which integral operators are of special interest in this application and develop methods for approximating them using the above result.

Develop local decompositions of Fourier matrices and/or circulant templates using Winograd's FFT algorithm [6, 66]. Parlett has expressed this FFT in terms of eigenvalue-eigenvector decompositions of circulant matrices [38]. Perhaps this fact can be of use.

Generalize the methods used in Chapter 4 to other families of matrices that are somehow "built up" using Kronecker products. The closest relative to the discrete Fourier transform having this property is the Hadamard transform. Hadamard matrices having dimensions a power of two can be defined in terms of Kronecker Products similar to the DFT. In fact, they can be considered to be multidimensional DFTs of length two in each dimension. In any event, it is immediate that these matrices have local decompositions similar to the DFT in the radix-two case. What happens in the general case? Is there an analogue to the Rader prime algorithm for Hadamard matrices? Answer these questions for more general families. From a different perspective, develop efficient methods for determining if a given matrix can be factored into a Kronecker product of two matrices, and, if so, for computing the decomposition. Such a decomposition would reduce the amount of computation required to implement the linear transform corresponding to that matrix.

Expand on the results in Chapter 5. The existence theorems of Chapter 2 imply that every square matrix can be factored into a product of tridiagonal matrices. Use methods other than the minimal variable method to solve the system of nonlinear equations that need to be satisfied for oblique elimination to be successful. Characterize the class, or classes, of matrices for which these methods will be successful. The method developed by Tchente will not work on every matrix and the minimal variable oblique algorithm is even more restrictive. Moreover, tridiagonal decompositions obtained using oblique elimination of any sort results in bidiagonal rather than tridiagonal factors. Develop entirely different methods for factoring square matrices into products of tridiagonals. We know of no other work in the area and communication with experts in the field of numerical linear algebra, such as Professor G. H. Golub of Stanford University, suggest that the problem of factoring arbitrary square matrices into products of tridiagonal matrices is as yet unsolved. More generally, develop methods for factoring linear transformations into products of local transformations relative to other configurations that model parallel computer architectures. Again, by the existence theorem of Chapter 2, this can be done for a large class of configurations. Use the concepts of Fourier analysis and synthesis in group representation theory to develop methods for factoring G-templates into products of templates local with respect to Cayley networks.

APPENDIX COMPUTER PROGRAMS

In this appendix, we present computer programs used to implement some local algorithms for computing DFT's using local decompositions of Fourier matrices derived in chapters 4 and 5. We include these programs for two reasons: One reason is so that the programs can be duplicated by someone else who may be interested to see the algorithms work and the other is to illustrate the use of the image algebra presented in this dissertation as a basis for an algebraically based, parallel programming language. These programs were written and run on a VAX 11/750 running on the 4.2 BSD UNIX operating system. The first two programs are written in FORTRAN 77 and are used to compute minimal variable decompositions of Fourier matrices and to implement the parallel algorithms resulting from these decompositions. We then describe two programs which were written using an extension of FORTRAN 77 which provides for the use of image algebra operands and operators. One of these programs will compute the DFT of a 100×100 image using the FFT-based method whereas the other is a hybrid program using both the FFT-based method and oblique elimination. We point out that, since these programs were run on a serial machine, they are not truly parallel programs. The programs written using the extended image algebra FORTRAN is written in parallel fashion since it uses the image algebra. The program for computing tridiagonal decompositions of Fourier matrices by minimal variable oblique elimination is not meant to be parallel. The program for implementing the parallel algorithms resulting from these decompositions is written in a serial fashion but can easily be made parallel. It illustrates the ease

with which a program for computing the DFT of sequences of arbitrary length locally can be written. The hybrid program mentioned earlier contains a segment of code which is a parallel implementation of the local algorithm for computing 5-point DFT's resulting from minimal variable oblique elimination.

FORTRAN Programs for Minimal Variable Oblique Elimination

In this section, we present two programs, `coeffgen.f` and `oem.f`. The programs `coeffgen.f` takes as input an integer $n > 1$ and generates the coefficients and permutation information corresponding to the minimal variable decomposition of F_n . A description of the program steps follows:

1.) Input n .

2.) Generate F_n .

3.) Compute LU decomposition of PF_n for some permutation matrix P using Gaussian elimination with partial pivoting; that is, compute the decomposition $F_n = PLU$ where P is a permutation matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix.

4.) Use minimal variable oblique elimination to compute the decompositions

$$L = Y_1 Y_2 \cdots Y_{n-2} B$$

and

$$U = C X_{n-2} X_{n-3} \cdots X_1$$

where, for $i \in [1, n-2]$, the matrices Y_i and X_i are unit lower and upper bidiagonal, respectively, and B and C are lower and upper bidiagonal, respectively. In the program, the sub-diagonal elements of the matrices Y_1, Y_2, \dots , and Y_n are stored in the first $n-2$ rows of the two-dimensional array $Y(i,j)$ and the sub-diagonal and diagonal elements of

the matrix B are stored in the last two rows of $Y(i,j)$. Similarly, the super-diagonal elements of the matrices X_1, X_2, \dots , and X_n are stored in the first $n-2$ rows of the two-dimensional array $X(i,j)$ and the super-diagonal and diagonal elements of the matrix C are stored in the last two rows of $X(i,j)$. These are the coefficients that we refer to and they are written to the files `xcoeffn` and `ycoeffn`.

4.) Generate a table `oempermn` containing information about the permutation used to implement the partial pivoting. An array R is constructed which contains this information and which is written to the file `oempermn`. The array R is constructed according to the following rule:

do $i=1,n$

If row i is switched with row j during the partial pivoting, then $R(i)=j$.

Thus, R contains a sequential record of the transpositions used to implement partial pivoting. The permutation matrix P in 3.) represents the product of the inverse of these transpositions and is implemented accordingly. For example, suppose $n = 5$ and first row 2 is switched with row 3 and then row 3 is switched with row 4 and the other rows remain fixed. Then P represents the permutation $\sigma = ((23)(34))^{-1} = (34)(23)$. Thus, P can be implemented by starting through an array of data of length 5 at the fifth location and sequentially making the transpositions as necessary. Since we are using a serial machine, this is the method that we use rather than the OETS which is very time consuming on such a machine.

The program `coeffgen.f` follows:

```

C   Program coeffgen.f
      complex A(100,100),B(100,100),C(100,100)
      complex X(100,100),Y(100,100)
      integer R(100)
      complex wn,z,d
      character yfile*9,xfile*9,blocksize*3,permfile*10

```

```

C*****
C
C   This program will generate the coefficients necessary to
C   compute the one-dimensional DFT of blocksize n locally
C   using oblique elimination. It will also generate the permutations.
C   The coefficients are written to the files xcoeffn and
C   ycoeffn and the permutations are written to the file oempermn.
C
C*****

```

```

      print *, "Enter the character representation of the blocksize,"
      print *, "that is, enter the blocksize enclosed in quotes."
      read *, blocksize
      xfile='xcoeff'//blocksize
      yfile='ycoeff'//blocksize
      permfile='oemperm'//blocksize

```

```

      print *, "Enter numerical blocksize."
      read *, n
      print *, "blocksize =", n

```

```

C   The Fourier matrix of order n is generated here

```

```

      pi=3.14159265
      pn=2*pi/n
      wn=cmlpx(cos(pn),-sin(pn))

```

```

      do 101 i=1,n
        do 102 j=1,n
          A(i,j)=wn**((i-1)*(j-1))

```

```

102    continue
101    continue

```

```

C      The LU decomposition of the Fourier matrix is computed here

do 105 i=1,n-1
  B(i,i)=(1.0,0.0)

C      The partial pivoting is done here.
  nmax=i
  do 201 np=i+1,n
    If (abs(A(np,i)) .gt. abs(A(nmax,i))) Then
      nmax=np
    endif
201  continue
  R(i)=nmax
  If (nmax .ne. i ) then
    do 202 ns=i,n
      z = A(i,ns)
      A(i,ns)=A(nmax,ns)
      A(nmax,ns)=z
202  continue
    endif
    do 106 k=i+1,n
      If ( A(i,i) .EQ. (0.0,0.0)) then
        print *, " at i= ",i
        print *, "Ran into zero divide using Gaussian Elimination."
        stop
      else
        B(k,i) = A(k,i)/A(i,i)
      endif
    do 107 j=i+1,n
      A(k,j)=A(k,j)-B(k,i)*A(i,j)
107  continue
106  continue
105  continue
  R(n)=n
  C(n,n)=(1.0,0.0)

```

C The permutations of the lower triangular required to compensate
 C for the partial pivoting are performed here.

```

do 650 i=n-1,1,-1
do 651 j=i,n
  C(j,i)=B(j,i)
651 continue
  if (R(i) .ne. i) then
    do 652 j=1,n
      z=C(R(i),j)
      C(R(i),j)=C(i,j)
      C(i,j)=z
652 continue
    endif
650 continue

do 655 i=1,n-1
  if (R(i) .ne. i) then
    do 656 j=1,n
      z=C(R(i),j)
      C(R(i),j)=C(i,j)
      C(i,j)=z
656 continue
    endif
655 continue
```

C We set the subdiagonals of the upper triangular factor
 C equal to zero here. This is necessary to avoid errors
 C during the oblique elimination.

```

do 150 i=2,n
do 150 j=1,i-1
  A(i,j)=(0.0,0.0)
150 continue
```

```

C      The coefficients are calculated here. That is the oblique elimination
C      is performed on the upper and lower triangular factors of the Fourier
C      matrix

      do 99 ns=1,2

      do 20 L=1,n-2
        do 5 j=1,n-1-L
          X(L,j)=cmplx(0.0,0.0)
5          continue

          do 10 k=n-L,n-1
            d=C(k,k+1-(n-L))
            j=1
            z=cmplx(1.0,0.0)
15          If (X(L,k-j) .NE. cmplx(0.0,0.0)) then
            z=z*X(L,k-j)
            d=d+C(k-j,k+1-(n-L))*z

            j=j+1
            go to 15
          endif
          If ( d .EQ. cmplx(0.0,0.0) ) then
            print *, "ran into zero divide"
            stop
          endif
          X(L,k)=-C(k+1,k+1-(n-L))/d
10          continue
      call ltmult(X,C,L,n)
20      continue

```

```

C      At this point the coefficients of the last tridiag are put in the
C      last two rows of the coefficient array

      X(n-1,1)=C(1,1)
      do 601 j=2,n
         X(n-1,j)=C(j,j)
         X(n,j)=C(j,j-1)
601      continue

      If (ns .eq. 1 ) then

C      At this point the coefficients are put into another array for storage
C      while the coefficients of the upper tridiagonal are computed

      do 603 i=1,n
      do 602 j=1,n
         Y(i,j)=X(i,j)
602      continue
603      continue

C      The upper triangular matrix is taken out of storage so that it
C      can be factored

      do 501 i=1,n
      do 502 j=1,n
         C(i,j)=A(j,i)
502      continue
501      continue
      endif

99      continue

```

C The coefficients are written to a file here.

```

open(15,file=xfile,status='new')
do 902 i=1,n
    do 901 j=1,n
        write(15,*) X(i,j)
901    continue
902    continue
close(15)

open(15,file=yfile,status='new')
do 904 i=1,n
    do 903 j=1,n
        write(15,*) Y(i,j)
903    continue
904    continue
close(15)

open(15,file=permfile,status='new')
do 950 i=1,n
    write(15,*) R(i)
950    close(15)

end

```



```

C *****
  subroutine ltmult(X,B,L,n)
  complex X(100,100),B(100,100),C(100,100)
  complex y

C*****
C
C   This subroutine performs the matrix multiplication necessary to
C   eliminate the L th oblique of the lower banded n x n matrix B.
C   B has zeroes in the 1,2,...,L-1 lower obliques on input and
C   zeroes in the 1,2,...,L lower obliques on output. The coefficients
C   required to perform this elimination are stored in the array X
C   and have been computed in the main program.
C
C*****

      do 25 i=1,n
      do 26 j=1,n
      c(i,j)=(0.0,0.0)
26      continue
25      continue

      C(1,1)=B(1,1)
      do 30 i=2,n
      y=(1.0,0.0)
      do 35 k=i,2,-1
      if ( k .NE. i ) then
      y=y*X(L,k)
      endif
      do 40 j=1,i
      C(i,j)=C(i,j)+y*B(k,j)
40      continue
35      continue
30      continue

      do 50 ii=1,n
      do 55 jj=1,n
      B(ii,jj)=C(ii,jj)
55      continue
50      continue
      end

```

The program oem.f uses the information generated by coeffgen.f to compute the one-dimensional DFT of a sequence of length n locally. The following steps are executed by the program:

1.) Input n , filename of file containing input sequence, and filename of file to write output sequence to,

2.) Read from coefficient and permutation files generated by coeffgen.f,

3.) Read input sequence \vec{v} ,

4.) Perform the following sequence of calculations:

do $i=1, n-2$

$\vec{v} \leftarrow X_i \vec{v}$,

5.) Perform $\vec{v} \leftarrow C\vec{v}$,

6.) Perform $\vec{v} \leftarrow B\vec{v}$,

7.) Perform the following sequence of calculations:

do $i=n-2, 1$

$\vec{v} \leftarrow Y_i \vec{v}$,

8.) Perform $\vec{v} \leftarrow P\vec{v}$,

9.) Write output sequence to a file,

where $F_n = PY_1Y_2 \cdots BCX_{n-2}X_{n-3} \cdots X_1$ is the minimal variable decomposition of F_n .

The program oem.f follows:

C Program oem.f

```
complex X(100,100),Y(100,100),V(100),W(100)
integer R(100)
complex wn,z
character xfile*9,yfile*9,permfile*10,blocksize*3
character output*10,input*10
```

C*****

C This program will compute an n-point DFT locally using
C the decompositions computed using oblique elimination.
C The program reads the coefficients from the files xcoeff
C and ycoeff and the permutations from the file oempermn.
C It reads the input sequence from the user defined file
C input and writes the output sequence to the user defined
C file output.

C*****

C The coefficient and permutation input filenames are constructed here.

```
print *,"Enter the character representation of the blocksize,"
print *,"that is, enter the blocksize enclosed in quotes."
read *,blocksize
xfile='xcoeff'//blocksize
yfile='ycoeff'//blocksize
permfile='oemperm'//blocksize
print *,"Enter the numerical representation of the blocksize."
read *,n
```

C The input and output filenames are input here.

```
print *,"Enter the input filename (enclosed in quotes)."
```

```
read *,input
```

```
print *,"Enter the output filename (enclosed in quotes)."
```

```
print *,"It must be a new file."
```

```
read *,output
```

- C The coefficients and permutation information are input here.

```

open(15,file=xfile,status='old')
do 1 i=1,n
  do 2 j=1,n
    read(15,*) X(i,j)
2    continue
1  continue
close(15)

open(15,file=yfile,status='old')
do 3 i=1,n
  do 4 j=1,n
    read(15,*) Y(i,j)
4    continue
3  continue
close(15)

open(15,file=permfile,status='old')
do 5 i=1,n
  read(15,*) R(i)
5  continue
close(15)

```

- C The input sequence is obtained here.

```

open(15,file=input,status='old')
do 6 i=1,n
  read(15,*) V(i)
6  continue
close(15)

```

C The DFT of the array V is now computed locally using the
 C coefficients derived from the oblique elimination process.

```

do 7 i=1,n-2
  do 8 j=1,n-1
    W(j)=V(j)-X(i,j)*V(j+1)
8    continue
  do 9 j=1,n-1
    V(j)=W(j)
9    continue
7    continue

do 10 j=1,n-1
  W(j)=X(n-1,j)*V(j)+X(n,j+1)*V(j+1)
10  continue
  V(n)=X(n-1,n)*V(n)
do 11 j=1,n-1
  V(j)=W(j)
11  continue

do 12 j=n,2,-1
  W(j)=Y(n,j)*V(j-1)+Y(n-1,j)*V(j)
12  continue
do 13 j=2,n
  V(j)=W(j)
13  continue
  V(1)=Y(n-1,1)*V(1)

do 14 i=n-2,1,-1
  do 15 j=n,2,-1
    W(j)=V(j)-Y(i,j-1)*V(j-1)
15    continue
  do 16 j=1,n
    V(j)=W(j)
16    continue
14    continue

do 17 j=1,n
  V(j)=(1/cmplx(n))*V(j)
17  continue

```

C The permutations corresponding to the partial
 C pivots are executed here. Since the permutations are generated
 C from the partial pivoting process, they are represented as a
 C sequence of transpositions corresponding to the switching
 C of the rows.

```

do 18 i=n,1,-1
  if (R(i) .ne. i) then
    z=V(i)
    V(i)=V(R(i))
    V(R(i))=z
  endif
18 continue

```

C The output sequence is now written to a file.

```

open(15,file=output,status='new')
do 19 i=1,n
19   write(15,*) V(i)
close(15)

end

```

These programs have been run and tested against various existing DFT and FFT programs. The results indicate that round-off error becomes increasingly significant when computing minimal variable decompositions of Fourier matrices. When n gets close to 30, the computations “blow-up”. Use of double precision alleviates the problem somewhat. A detailed analysis of this situation is needed. We point out that the decompositions only need to be computed once. Thus, it is not unreasonable to consider developing a very precise, but possibly slow, algorithm for generating the coefficients required for implementing the DFT, or some other particular linear transform, locally using minimal variable oblique elimination.

Image Algebra Programs for the 100 x 100 DFT

The next two programs that we describe are written in an extended FORTRAN 77 code made available by an image algebra preprocessor. The image algebra preprocessor is a FORTRAN 77 program which allows for programs to be written using image algebra operands and operators. The operands and operators of the image algebra are representable in an extended FORTRAN 77 code which is input to the preprocessor. The preprocessor scans the code for special symbols corresponding to image algebra objects. The lines containing image algebra statements are replaced by equivalent blocks of FORTRAN 77 code. The result is a FORTRAN 77 program which can be compiled and run in the usual way. The preprocessor is for experimental use only and is a straightforward implementation with few special features or optimization techniques included; we describe it only as much as is needed to understand the code that we present.

Images are limited to two-dimensional arrays and are declared as such in the usual way. Templates are constructed in a fashion similar to subroutines and can be defined with parameters, in fact, they are implemented as subroutines. They are declared at the beginning of the main program and can either be defined internally after the end of the main program or they can be defined externally and be stored in a template library. A template is defined by defining the weights at some set of points. This set of points is taken to be the configuration. The operations of $+$ and $*$ between images are represented by these symbols for the preprocessor. The operation of \oplus between images and templates is represented by the symbol $+$. The preprocessor is "aware" of the types of objects that it is dealing with and interprets the $+$ sign accordingly. As an example, suppose that $\mathbf{X} = \{(x,y) : 0 \leq x \leq 99, 0 \leq y \leq 99\}$ is a 100 x 100 array and that one wants to define the templates $(T, T), (S, S) \in L_{\mathbf{X}}$ defined by:

T:

If y is even then for every x

$$t_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y+1). \\ 0 & \text{else} \end{cases}$$

If y is odd then for every x

$$t_{(x,y)}(u,v) = \begin{cases} -1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x,y-1). \\ 0 & \text{else} \end{cases}$$

S:

If x is even then for every y

$$s_{(x,y)}(u,v) = \begin{cases} 1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x+1,y). \\ 0 & \text{else} \end{cases}$$

If x is odd then for every y

$$s_{(x,y)}(u,v) = \begin{cases} -1 & \text{if } (u,v) = (x,y) \\ 1 & \text{if } (u,v) = (x-1,y). \\ 0 & \text{else} \end{cases}$$

The templates T and S are used to implement the transformation corresponding to the matrix $I_{50} \otimes F_2$ along each row and each column, respectively, of a 100 x 100 image. This type of computation is sometimes called a *butterfly computation*. An example preprocessor program in which these templates are combined into one using a template with parameter follows:


```

complex variant bfly
complex A(100,100),B(100,100)

A=100
B=A+bfly(1)
end

complex variant template bfly(nrc)

if (nrc .eq. 1 ) then

  ybar=mod(y-1,2)
  if (ybar .eq. 0) then
    bfly(x,y)=cmplx(1)
    bfly(x,y+1) =cmplx(1)
  else
    bfly(x,y)=cmplx(-1)
    bfly(x,y-1)=cmplx(1)
  endif
endif

else

  xbar=mod(x-1,2)
  if (xbar .eq. 0) then
    bfly(x,y)=cmplx(1)
    bfly(x+1,y) =cmplx(1)
  else
    bfly(x,y)=cmplx(-1)
    bfly(x-1,y)=cmplx(1)
  endif
endif
end

```

The statement $A=100$ will initialize every location of A to the value 100, that is, it is a parallel assignment statement. The statement $B = A + \text{bfly}(1)$ represents $B = A \oplus T$. The first image algebra program that we describe, `dft100dp.f85`, takes as input a 100×100 image, A , and produces as output the DFT of A . The DFT is computed using an image algebra implementation of the twiddle free FFT-based decompositions of the Fourier matrices. The addition steps are all implemented locally using \oplus between images and templates. The multiplication steps are all implemented using image multi-

plication. In the interest of saving some time, the permutations are implemented directly rather than using the OETS.

Specifically, using the notation of Chapter 4, the decomposition implemented is the following:

$$\begin{aligned} F_{100} &= Q(4,25,3)(F_4 \otimes I_{25})(I_4 \otimes F_{25})(T_4(C_{25}^{19}))P(25,4) = \\ &= P(25,4)T_{25}(C_4^3)(I_{25} \otimes F_4)P(4,25) \\ &\quad (I_4 \otimes P(5,5))(I_{20} \otimes F_5)(I_4 \otimes P(5,5))(I_4 \otimes D(5,5))(I_{20} \otimes F_5)(I_4 \otimes P(5,5))(T_4(C_{25}^{19}))P(25,4). \end{aligned}$$

By Theorem 4.12,

$$F_5 = U_5 R_{1,5}(2) F_4^{(1)} \Lambda_5 F_4^{*(1)} R_{2,5}(2) U_5^t$$

and, by Theorem 4.13,

$$U_5 = V_4 V_3 V_2 V.$$

Furthermore,

$$I_{25} \otimes F_4 = (I_{25} \otimes P(2,2))(I_{25} \otimes F_2)(I_{25} \otimes P(2,2))(I_{25} \otimes D(2,2))(I_{25} \otimes F_2)(I_{25} \otimes P(2,2)).$$

Thus, if we make the substitutions

$$R_1 \equiv R_{1,5}(2),$$

$$R_2 \equiv R_{2,5}(2),$$

$$P6 \equiv T_4(C_{25}^{19})P(25,4),$$

$$P5 \equiv (I_4 \otimes P(5,5)),$$

$$P4 \equiv (I_{20} \otimes R_2)(I_{20} \otimes P(2,2)^t),$$

$$P3 \equiv (I_{20} \otimes P(2,2)^t)(I_{20} \otimes R_1),$$

$$P2 \equiv (I_{25} \otimes P(2,2))P(4,25)(I_4 \otimes P(5,5)),$$

and

$$P1 \equiv P(25,4)T_{25}(C_4^3)(I_{25} \otimes P(2,2)),$$

the program dft100dp.f85 represents the implementation of the sequence of local transformations indicated in Table A.1 along each row of the input image and then along each column of the result.

Table A.1. Sequence of Local Transformations Used to Implement
100 x 100 DFT Using Pure FFT-Based Algorithm.

Matrix	Type	# of Parallel Steps	Operations/Pixel/Step
1.) P6	P	82	
2.) P5	P	17	
3.) $I_{20} \otimes V_4^t$	A	1	1
4.) $I_{20} \otimes V_3^t$	A	1	1
5.) $I_{20} \otimes V_2^t$	A	1	1
6.) $I_{20} \otimes V^t$	A	1	1
7.) P4	P	5	
8.) $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$	A	1	1
9.) $I_{20} \otimes P(2,2)^{(1)}$	P	1	
10.) $I_{20} \otimes D(2,2)^{* (1)}$	M	1	1
11.) $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$	A	1	1
12.) $I_{20} \otimes P(2,2)^{(1)}$	P	1	
13.) $I_{20} \otimes \Lambda_5$	M	1	1
14.) $I_{20} \otimes P(2,2)^{(1)}$	P	1	
15.) $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$	A	1	1
16.) $I_{20} \otimes D(2,2)^{(1)}$	M	1	1
17.) $I_{20} \otimes P(2,2)^{(1)}$	P	1	
18.) $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$	A	1	1
19.) P3	P	5	
20.) $I_{20} \otimes V$	A	1	1
21.) $I_{20} \otimes V_2$	A	1	1
22.) $I_{20} \otimes V_3$	A	1	1
23.) $I_{20} \otimes V_4$	A	1	1
24.) $I_4 \otimes D(5,5)$	M	1	1
Repeat steps 2.) through 23.)			
25.) P2	P	100	
26.) $I_{50} \otimes F_2$	A	1	1
27.) $I_{25} \otimes D(2,2)$	M	1	1
28.) $I_{25} \otimes P(2,2)$	P	1	
29.) $I_{50} \otimes F_2$	A	1	1
30.) P1	P	100	

M stands for multiplication, A for addition, and P for permutation.

Note that it would take, at most, 690 parallel permutation steps, 18 parallel multiplication steps, and 52 parallel addition steps if this program were implemented on a parallel machine. Since they require a significant amount of floating point operations, some of the images used by the program to perform the multiplications, specifically, the images corresponding to the matrices $I_{20} \otimes \Lambda_5$ and $I_4 \otimes D(5,5)$, are generated and stored in files to be read as input to the program `dft100dp.f`. The other multiplication images are defined internal to the program since they can be defined without floating point operations. The permutations P1,P2,P3,P4,P5,P6 of Table A.1 are also pre-generated and stored in files. They are implemented using a general purpose subroutine. Due to the simplicity of their structure, the other permutations are implemented directly. Many of the template definitions involve invoking a call to the `mod` function. Thus, two tables are generated at the beginning of the program in order to save some time. The program `dft100dp.f85` executes the following steps:

- 1.) Read in input image, permutation tables, and multiplication images.

- 2.) Construct multiplication images.

- 3.) Execute sequence of local transformations given in Table A.1 twice, first along the rows and then along the columns of the result. The same templates are used for both, the multiplication images are transposed.

- 4.) Output the DFT of the input image.

To save time, the transposition is not performed locally. Jesshope has considered methods for transposing images on mesh-connected arrays [21]. It may be easier to rebroadcast the information to the individual processors, however. The main program follows:

C Program dft100dp.f85

```

common m2, m5
integer m2(0:99), m5(0:99)
complex variant bfly,ebfly,v2,v2t,v3,v3t,v4,v4t,v,vt,trans
complex A(100,100),B(100,100)
complex D2CT(100,100),D22(100,100),D221(100,100)
complex D55(100,100),L5A(100,100)
integer p6a(100),p5a(100),p4a(100),p3a(100),p2a(100),p1a(100)

call makemods()

```

C Reading in image

```

open(unit=15,file='testim',status='old')
do 1,i=1,100
1   read(15,200)(A(j,i),j=1,100)
200 format(200(F8.3,1X))
close(15)

```

C reading in permutation tables

```

open(unit=15,file='P1',status='old')
read(15,*)(p1a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P2',status='old')
read(15,*)(p2a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P3',status='old')
read(15,*)(p3a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P4',status='old')
read(15,*)(p4a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P5',status='old')
read(15,*)(p5a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P6',status='old')
read(15,*)(p6a(j),j=1,100)
close(15)

```

C reading in multiplication image lambda5

```

open(unit=15,file='L5dp',status='old')
do 7 j=1,100
  read(15,*) L5A(1,j)
  do 8 n=1,100
8    L5A(n,j)=L5A(1,j)
7  continue
close(15)

```

C reading in twiddle factor I5 tensor D(5,5)

```

open(unit=15,file='twiddledp',status='old')
do 15 j=1,100
  read(15,*) D55(1,j)
  do 16 n=1,100
16    D55(n,j)=D55(1,j)
15  continue
close(15)

```

C defining the twiddle factors

```

D22=cmplx(1)
jbar=4
do 9 j=1,25
  do 10 n=1,100
10    D22(n,jbar)=(0,-1)
  jbar=jbar+4
9  continue

D221=cmplx(1)
jbar=5
do 11 j=1,20
  do 12 n=1,100
12    D221(n,jbar)=(0,-1)
  jbar=jbar+5
11  continue

D2CT=cmplx(1)
jbar=5
do 13 j=1,20
  do 14 n=1,100
14    D2CT(n,jbar)=(0,1)
  jbar=jbar+5
13  continue

```

C The computations begin here

```

do 100 k=1,2
  call perm(A,B,p6a,k)
  L=L+1
101  if ( L .lt. 3 ) then

        call perm(B,A,p5a,k)

```

C The computation of the 5-point transforms is begins here.

```

B=A+v4t(k)
A=B+v3t(k)
B=A+v2t(k)
A=B+vt(k)
call perm(A,B,p4a,k)
A=B+ebfly(k)
call p221(A,B,k)
A=B*D2CT
B=A+ebfly(k)
call p221(B,A,k)
B=A*L5A
call p221(B,A,k)
B=A+ebfly(k)
A=B*D221
call p221(A,B,k)
A=B+ebfly(k)
call perm(A,B,p3a,k)
A=B+v(k)
B=A+v2(k)
A=B+v3(k)
B=A+v4(k)

```

C The computation of the 5-point transforms ends here.

```

if ( L .eq. 2 ) go to 102
A=B*D55
B=A
L=L+1
go to 101
endif
102 continue

```

```

    call perm(B,A,p2a,k)

C    The computation of the 4-point transforms begins here.

    B=A+bfly(k)
    A=B*D22*0.01
    call p22(A,B,k)
    A=B+bfly(k)

C    The computation of the 4-point transforms ends here.
    call perm(A,B,p1a,k)

C    The multiplication images are transposed here.

    if (k .eq. 1) then
        A=B
        B=D2CT+trans
        D2CT=B
        B=L5A+trans
        L5A=B
        B=D221+trans
        D221=B
        B=D55+trans
        D55=B
        B=D22+trans
        D22=B
    endif

100    continue

C    The computations end here.

    open(unit=15,file='alg.out',status='new')
        do 400 j=1,100
            write(15,*) B(j,1)
400        continue
    close(15)

end

```


The templates and subroutines used by dft100dp.f85 are the following:

```
complex variant template bfly(nrc)
common  m2, m5
integer m2(0:99), m5(0:99)
```

C This is the template corresponding to $I_{50} \otimes F_2$.

```
if (nrc .eq. 1 ) then

ybar=m2(y-1)
if (ybar .eq. 0) then
  bfly(x,y)=cmplx(1)
  bfly(x,y+1) =cmplx(1)
else
  bfly(x,y)=cmplx(-1)
  bfly(x,y-1)=cmplx(1)
endif

else

xbar=m2(x-1)
if (xbar .eq. 0) then
  bfly(x,y)=cmplx(1)
  bfly(x+1,y) =cmplx(1)
else
  bfly(x,y)=cmplx(-1)
  bfly(x-1,y)=cmplx(1)
endif
endif
end
```

```

complex variant template ebfly(nrc)
common m2, m5
integer m2(0:99), m5(0:99)

```

C This is the template corresponding to $I_{20} \otimes (I_2 \otimes F_2)^{(1)}$.

```

if (nrc .eq. 1 ) then

  ybar=m5(y-1)
  if (ybar .eq. 0) then
    ebfly(x,y)=cmplx(1)
  else
    if (ybar .eq. 1 .or. ybar .eq. 3) then
      ebfly(x,y)=cmplx(1)
      ebfly(x,y+1)=cmplx(1)
    else
      if (ybar .eq. 2 .or. ybar .eq. 4) then
        ebfly(x,y)=cmplx(-1)
        ebfly(x,y-1)=cmplx(1)
      endif
    endif
  endif

else

  xbar=m5(x-1)
  if (xbar .eq. 0) then
    ebfly(x,y)=cmplx(1)
  else
    if (xbar .eq. 1 .or. xbar .eq. 3) then
      ebfly(x,y)=cmplx(1)
      ebfly(x+1,y)=cmplx(1)
    else
      if (xbar .eq. 2 .or. xbar .eq. 4) then
        ebfly(x,y)=cmplx(-1)
        ebfly(x-1,y)=cmplx(1)
      endif
    endif
  endif
endif

end

```

The next eight templates correspond to the tridiagonal decomposition of U_5 .

```
complex variant template v2(nrc)
```

```
common m2, m5
```

```
integer m2(0:99), m5(0:99)
```

```
if (nrc .eq. 1 ) then
```

```
  ybar=m5(y-1)
```

```
  if (ybar .eq. 2) then
```

```
    v2(x,y)=cmplx(1)
```

```
    v2(x,y-1) =cmplx(1)
```

```
  else
```

```
    v2(x,y)=cmplx(1)
```

```
  endif
```

```
else
```

```
  xbar=m5(x-1)
```

```
  if (xbar .eq. 2) then
```

```
    v2(x,y)=cmplx(1)
```

```
    v2(x-1,y) =cmplx(1)
```

```
  else
```

```
    v2(x,y)=cmplx(1)
```

```
  endif
```

```
endif
```

```
end
```

```
complex variant template v2t(nrc)
```

```
common m2, m5
```

```
integer m2(0:99), m5(0:99)
```

```
if (nrc .eq. 1 ) then
```

```
  ybar=m5(y-1)
```

```
  if (ybar .eq. 1) then
```

```
    v2t(x,y)=cmplx(1)
```

```
    v2t(x,y+1) =cmplx(1)
```

```
  else
```

```
    v2t(x,y)=cmplx(1)
```

```
  endif
```

```
else
```

```
  xbar=m5(x-1)
```

```
  if (xbar .eq. 1) then
```

```
    v2t(x,y)=cmplx(1)
```

```
    v2t(x+1,y) =cmplx(1)
```

```

else
  v2t(x,y)=cmplx(1)
endif
endif
end

complex variant template v3(nrc)
common m2, m5
integer m2(0:99), m5(0:99)

if (nrc .eq. 1 ) then

  ybar=m5(y-1)
  if (ybar .eq. 3) then
    v3(x,y)=cmplx(1)
    v3(x,y-1) =cmplx(1)
  else
    v3(x,y)=cmplx(1)
  endif

else

  xbar=m5(x-1)
  if (xbar .eq. 3) then
    v3(x,y)=cmplx(1)
    v3(x-1,y) =cmplx(1)
  else
    v3(x,y)=cmplx(1)
  endif
endif
end

complex variant template v3t(nrc)
common m2, m5
integer m2(0:99), m5(0:99)

if (nrc .eq. 1 ) then

  ybar=m5(y-1)
  if (ybar .eq. 2) then
    v3t(x,y)=cmplx(1)
    v3t(x,y+1) =cmplx(1)
  else
    v3t(x,y)=cmplx(1)
  endif

else

  xbar=m5(x-1)

```

```

if (xbar .eq. 2) then
  v3t(x,y)=cmplx(1)
  v3t(x+1,y) =cmplx(1)
else
  v3t(x,y)=cmplx(1)
endif
endif
end

```

complex variant template v4(nrc)

```

common m2, m5
integer m2(0:99), m5(0:99)

```

```

if (nrc .eq. 1 ) then

```

```

  ybar=m5(y-1)
  if (ybar .eq. 4) then
    v4(x,y)=cmplx(1)
    v4(x,y-1) =cmplx(1)
  else
    v4(x,y)=cmplx(1)
  endif
endif

```

```

else

```

```

  xbar=m5(x-1)
  if (xbar .eq. 4) then
    v4(x,y)=cmplx(1)
    v4(x-1,y) =cmplx(1)
  else
    v4(x,y)=cmplx(1)
  endif
endif
end

```

complex variant template v4t(nrc)

```

common m2, m5
integer m2(0:99), m5(0:99)

```

```

if (nrc .eq. 1 ) then

```

```

  ybar=m5(y-1)
  if (ybar .eq. 3) then
    v4t(x,y)=cmplx(1)
    v4t(x,y+1) =cmplx(1)
  else
    v4t(x,y)=cmplx(1)
  endif
endif

```

```

else

xbar=m5(x-1)
if (xbar .eq. 3) then
  v4t(x,y)=cmplx(1)
  v4t(x+1,y)=cmplx(1)
else
  v4t(x,y)=cmplx(1)
endif
endif
end

complex variant template v(nrc)
common m2, m5
integer m2(0:99), m5(0:99)

if (nrc .eq. 1 ) then

ybar=m5(y-1)
if (ybar .eq. 0) then
  v(x,y)=cmplx(1)
else
  if (ybar .eq. 1) then
    v(x,y)=cmplx(1)
    v(x,y-1)=cmplx(1)
  else
    v(x,y)=cmplx(1)
    v(x,y-1)=cmplx(-1)
  endif
endif

else

xbar=m5(x-1)
if (xbar .eq. 0) then
  v(x,y)=cmplx(1)
else
  if (xbar .eq. 1) then
    v(x,y)=cmplx(1)
    v(x-1,y)=cmplx(1)
  else
    v(x,y)=cmplx(1)
    v(x-1,y)=cmplx(-1)
  endif
endif
endif
end

```

```

complex variant template vt(nrc)
common  m2, m5
integer m2(0:99), m5(0:99)

```

```

if (nrc .eq. 1 ) then

```

```

  ybar=m5(y-1)
  if (ybar .eq. 4) then
    vt(x,y)=cmplx(1)
  else
    if (ybar .eq. 0) then
      vt(x,y)=cmplx(1)
      vt(x,y+1)=cmplx(1)
    else
      vt(x,y)=cmplx(1)
      vt(x,y+1)=cmplx(-1)
    endif
  endif
endif

```

```

else

```

```

  xbar=m5(x-1)
  if (xbar .eq. 4) then
    vt(x,y)=cmplx(1)
  else
    if (xbar .eq. 0) then
      vt(x,y)=cmplx(1)
      vt(x+1,y)=cmplx(1)
    else
      vt(x,y)=cmplx(1)
      vt(x+1,y)=cmplx(-1)
    endif
  endif
endif
endif
end

```

```

subroutine makemods()
common m2, m5
integer m2(0:99), m5(0:99)

```

C This is the subroutine to generate the mod tables.

```

do 10 i = 0, 99
    m2(i) = mod(i,2)
    m5(i) = mod(i,5)
10 continue
end

```

```

subroutine perm(A,B,P,rc)
complex A(100,100),B(100,100)
integer P(100),rc

```

C This is the general purpose permutation subroutine.

C A contains the image to be permuted.

C B will contain the output image.

C P contains the permutation to be implemented.

C rc=1 indicates permute along rows, rc=2 along columns.

```

do 1 i=1,100
do 1 j=1,100
    if ( rc .eq. 1 ) then
        B(i,j) = A(i,P(j))
    else
        B(i,j) = A(P(i),j)
    endif
1 continue
end

```



```

subroutine p221(A,B,nrc)
complex A(100,100),B(100,100)

```

- C This subroutine implements the permutation $I_{20} \otimes P(2,2)^{(1)}$.
 C A contains the image to be permuted.
 C B will contain the output image.
 C nrc=1 indicates permute along rows, nrc=2 along columns.

```

      jbar=3
      B=A
      do 1,j=1,20
      if (nrc .eq. 1) then
        do 2 n=1,100
          B(n,jbar)=A(n,jbar+1)
          B(n,jbar+1)=A(n,jbar)
2       continue
      else
        do 3 n=1,100
          B(jbar,n)=A(jbar+1,n)
          B(jbar+1,n)=A(jbar,n)
3       continue
      endif
      jbar=jbar+5
1     continue

      end

```

```
subroutine p22(A,B,nrc)
complex A(100,100),B(100,100)
```

- C This subroutine implements the permutation $I_{25} \otimes P(2,2)$.
 C A contains the image to be permuted.
 C B will contain the output image.
 C nrc=1 indicates permute along rows, nrc=2 along columns.

```

jbar=2
B=A
do 1,j=1,25
if (nrc .eq. 1) then
do 2 n=1,100
B(n,jbar)=A(n,jbar+1)
B(n,jbar+1)=A(n,jbar)
2 continue
else
do 3 n=1,100
B(jbar,n)=A(jbar+1,n)
B(jbar+1,n)=A(jbar,n)
3 continue
endif
jbar=jbar+4
1 continue

end

complex variant template trans
```

- C This template will transpose an image.

```
trans(y,x)=cmplx(1)
end
```

The next program, oem100.f85, is the same as dft100dp.f85 except that instead of using the Rader prime algorithm and the convolution theorem to compute the 5-point transforms, oblique elimination is used. Denote the minimal variable decomposition of F_5 by $F_5 = P_5 M_8 M_7 \cdots M_1$. Then, the program is an implementation of the sequence of transformations shown in Table A.2.

Table A.2. Sequence of Local Transformations Used to Implement
100 x 100 DFT Using Hybrid Algorithm.

Matrix	Type	# of Parallel Steps	Operations/Pixel/Step
1.) P_6	P	82	
2.) P_5	P	17	
3.) M_1	A/M	1	2
4.) M_2	A/M	1	2
5.) M_3	A/M	1	2
6.) M_4	A/M	1	3
7.) M_5	A/M	1	2
8.) M_6	A/M	1	2
9.) M_7	A/M	1	2
10.) M_8	A/M	1	2
11.) P_5	P	5	
11.) $I_4 \otimes D(5,5)$	M	1	1
Repeat steps 2.) through 10.)			
12.) P_2	P	100	
13.) $I_{50} \otimes F_2$	A	1	1
14.) $I_{25} \otimes D(2,2)$	M	1	1
15.) $I_{25} \otimes P(2,2)$	P	1	
16.) $I_{50} \otimes F_2$	A	1	1
17.) P_1	P	100	

M stands for multiplication, A for addition, and P for permutation.

Note that this implementation would take, at most, 654 parallel permutation steps, 36 multiplication steps, and 36 addition steps. Although these numbers do not appear to be as good as for the pure FFT-based decompositions, it is interesting to observe that

the program oem100.f85 runs significantly faster (about 30%) than dft100dp.f85 on the VAX. A possible reason for this is that the different types of steps are more interspersed in dft100dp.f85. Thus, there are more instructions necessary; in particular, there are more templates involved. Since each template involves a subroutine call for every point in the image every time that it is used, this is a significant point. The template definitions are also more straightforward. Finally, the preprocessor goes through the motions of multiplying by the weights in the \oplus operation, even if the weights are 1's. This seriously affects the performance of these programs since every template used in dft100dp.f85 has weights either 0 or 1. The templates t1,t2,...,t8 in the program correspond to the matrices M_1, M_2, \dots, M_8 , respectively. They were defined using the coefficients generated by the program coeffgen.f. The main program follows:

C Program oem100.f85

```

common m2, m5
integer m2(0:99), m5(0:99)
complex variant bfly,t1,t2,t3,t4,t5,t6,t7,t8,trans
complex A(100,100),B(100,100)
complex D22(100,100),D55(100,100)
integer p6a(100),p5a(100),p2a(100),p1a(100),oemp(100)

call makemods()

```

C Reading in image

```

open(unit=15,file='testim',status='old')
do 1,i=1,100
1   read(15,200)(A(i,j),j=1,100)
200 format(200(F8.3,1X))
close(15)

```

C Reading in permutation tables

```

open(unit=15,file='P1',status='old')
read(15,*)(p1a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P2',status='old')
read(15,*)(p2a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P5',status='old')
read(15,*)(p5a(j),j=1,100)
close(15)

```

```

open(unit=15,file='P6',status='old')
read(15,*)(p6a(j),j=1,100)
close(15)

```

```

open(unit=15,file='oemf85',status='old')
read(15,*)(oemp(j),j=1,100)
close(15)

```

```

C      reading in twiddle factor I5 tensor D(5,5)

      open(unit=15,file='twiddledp',status='old')
      do 15 j=1,100
      read(15,*) D55(1,j)
      do 16 n=1,100
16      D55(n,j)=D55(1,j)
15      continue
      close(15)

C      defining the twiddle factors

      D22=cmplx(1)
      jbar=4
      do 9 j=1,25
      do 10 n=1,100
10      D22(n,jbar)=(0,-1)
      jbar=jbar+4
9      continue

```

C The computations begin here

```

do 100 k=1,2
  call perm(A,B,p6a,k)
  L=L+1
101  if ( L .lt. 3 ) then
    call perm(B,A,p5a,k)

```

C The computation of the 5-point transforms begins here.

```

  B=A+t1(k)
  A=B+t2(k)
  B=A+t3(k)
  A=B+t4(k)
  B=A+t5(k)
  A=B+t6(k)
  B=A+t7(k)
  A=B+t8(k)
  call oemp(A,oemp,k)

```

C The computation of the 5-point transforms ends here.

```

  if ( L .eq. 2 ) go to 102
  B=A*D55
  L=L+1
  go to 101
endif
102 continue
  call perm(A,B,p2a,k)

```

C The computation of the 4-point transforms begins here.

```

  B=A+bfly(k)
  A=B*D22*0.01
  call p22(B,A,k)
  B=A+bfly(k)

```

C The computation of the 4-point transforms ends here.

```

  call perm(B,A,p1a,k)
  if (k .eq. 1) then
    B=D55+trans
    D55=B
    B=D22+trans
    D22=B
  endif
100 continue

```

C The computations end here.

```
open(unit=15,file='alg.out',status='new')
    do 400 j=1,100
        write(15,*) B(1,j)
    continue
400 close(15)

end
```


The templates and subroutines used by oem100.f85 are as follows:

```
complex variant template bfly(nrc)
common m2, m5
integer m2(0:99), m5(0:99)
```

- C This is the template corresponding to $I_{50} \otimes F_2$.

```
if (nrc .eq. 1 ) then

  ybar=m2(y-1)
  if (ybar .eq. 0) then
    bfly(x,y)=cmplx(1)
    bfly(x,y+1)=cmplx(1)
  else
    bfly(x,y)=cmplx(-1)
    bfly(x,y-1)=cmplx(1)
  endif
endif

else

  xbar=m2(x-1)
  if (xbar .eq. 0) then
    bfly(x,y)=cmplx(1)
    bfly(x+1,y)=cmplx(1)
  else
    bfly(x,y)=cmplx(-1)
    bfly(x-1,y)=cmplx(1)
  endif
endif
end
```

```
subroutine makemods()
common m2, m5
integer m2(0:99), m5(0:99)
```

- C This is the subroutine to generate the mod tables.

```
do 10 i = 0, 99
    m2(i) = mod(i,2)
    m5(i) = mod(i,5)
10 continue
end
```

```
subroutine perm(A,B,P,rc)
```

```
complex A(100,100),B(100,100)
```

```
integer P(100),rc
```

- C This is the general purpose permutation subroutine.
 C A contains the image to be permuted.
 C B will contain the output image.
 C P contains the permutation to be implemented.
 C rc=1 indicates permute along rows, rc=2 along columns.

```
do 1 i=1,100
```

```
do 1 j=1,100
```

```
if ( rc .eq. 1 ) then
```

```
  B(i,j) = A(i,P(j))
```

```
else
```

```
  B(i,j)= A(P(i),j)
```

```
endif
```

```
1 continue
```

```
end
```

```
subroutine p22(A,B,nrc)
```

```
complex A(100,100),B(100,100)
```

- C This subroutine implements the permutation $I_{25} \otimes P(2,2)$.
 C A contains the image to be permuted.
 C B will contain the output image.
 C nrc=1 indicates permute along rows, nrc=2 along columns.

```
jbar=2
```

```
B=A
```

```
do 1,j=1,25
```

```
if (nrc .eq. 1) then
```

```
do 2 n=1,100
```

```
  B(n,jbar)=A(n,jbar+1)
```

```
  B(n,jbar+1)=A(n,jbar)
```

```
2 continue
```

```
else
```

```
do 3 n=1,100
```

```
  B(jbar,n)=A(jbar+1,n)
```

```
  B(jbar+1,n)=A(jbar,n)
```

```
3 continue
```

```
endif
```

```
jbar=jbar+4
```

```
1 continue
```

```
end
```

complex variant template trans

- C This template will transpose an image.

```
trans(y,x)=cmlpx(1)
end
```

The next eight templates are the templates defined using minimal variable oblique elimination.

```
complex variant template t1(k)
common m2, m5
integer m2(0:99), m5(0:99)
```

```
t1(x,y)=(1.0,0.0)
```

```
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 3) then
    t1(x,y+1)=(1.0,0.0)
  endif
else
  xb=m5(x-1)
  if(xb .eq. 3) then
    t1(x+1,y)=(1.0,0.0)
  endif
endif
end
```

```
complex variant template t2(k)
common m2, m5
integer m2(0:99), m5(0:99)
```

```
t2(x,y)=(1.0,0.0)
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 2) then
    t2(x,y+1)=(1.0,0.0)
  endif
  if(yb .eq. 3) then
    t2(x,y+1)=(-0.809017,-0.587785)
  endif
else
  xb=m5(x-1)
  if(xb .eq. 2) then
    t2(x+1,y)=(1.0,0.0)
  endif
  if(xb .eq. 3) then
    t2(x+1,y)=(-0.809017,-0.587785)
  endif
end
```

```

endif
end

complex variant template t3(k)
common m2, m5
integer m2(0:99), m5(0:99)

t3(x,y)=(1.0,0.0)
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 1) then
    t3(x,y+1)=(1.0,0.0)
  endif
  if(yb .eq. 2) then
    t3(x,y+1)=(-0.809017,-0.587785)
  endif
  if(yb .eq. 3) then
    t3(x,y+1)=(-0.809017,0.587785)
  endif
endif
else
  xb=m5(x-1)
  if(xb .eq. 1) then
    t3(x+1,y)=(1.0,0.0)
  endif
  if(xb .eq. 2) then
    t3(x+1,y)=(-0.809017,-0.587785)
  endif
  if(xb .eq. 3) then
    t3(x+1,y)=(-0.809017,0.587785)
  endif
endif
end

```

```

complex variant template t4(k)
common m2, m5
integer m2(0:99), m5(0:99)

if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 0) then
    t4(x,y)=(1.0,0.0)
    t4(x,y+1)=(1.0,0.0)
  endif
  if(yb .eq. 1) then
    t4(x,y)=(-1.80902,-0.587785)
    t4(x,y+1)=(1.11803,1.53884)
  endif
  if(yb .eq. 2) then
    t4(x,y)=(-0.690983,-2.12663)
    t4(x,y+1)=(1.80902,1.31433)
  endif
  if(yb .eq. 3) then
    t4(x,y)=(-2.5,0.8123)
    t4(x,y+1)=(0.0,2.62866)
  endif
  if(yb .eq. 4) then
    t4(x,y)=(1.54508,-4.75528)
  endif
else
  xb=m5(x-1)
  if(xb .eq. 0) then
    t4(x,y)=(1.0,0.0)
    t4(x+1,y)=(1.0,0.0)
  endif
  if(xb .eq. 1) then
    t4(x,y)=(-1.80902,-0.587785)
    t4(x+1,y)=(1.11803,1.53884)
  endif
  if(xb .eq. 2) then
    t4(x,y)=(-0.690983,-2.12663)
    t4(x+1,y)=(1.80902,1.31433)
  endif
  if(xb .eq. 3) then
    t4(x,y)=(-2.5,0.8123)
    t4(x+1,y)=(0.0,2.62866)
  endif
  if(xb .eq. 4) then
    t4(x,y)=(1.54508,-4.75528)
  endif
endif
end

```

```

complex variant template t5(k)
common m2, m5
integer m2(0:99), m5(0:99)

t5(x,y)=(1.0,0.0)
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 1) then
    t5(x,y-1)=(1.0,0.0)
  endif
  if(yb .eq. 2) then
    t5(x,y-1)=(-0.190983,-0.587785)
  endif
  if(yb .eq. 3) then
    t5(x,y-1)=(0.809017,0.587785)
  endif
  if(yb .eq. 4) then
    t5(x,y-1)=(1.61803,0.0)
  endif
endif
else
  xb=m5(x-1)
  if(xb .eq. 1) then
    t5(x-1,y)=(1.0,0.0)
  endif
  if(xb .eq. 2) then
    t5(x-1,y)=(-0.190983,-0.587785)
  endif
  if(xb .eq. 3) then
    t5(x-1,y)=(0.809017,0.587785)
  endif
  if(xb .eq. 4) then
    t5(x-1,y)=(1.61803,0.0)
  endif
endif
endif
end

```

```

complex variant template t6(k)
common m2, m5
integer m2(0:99), m5(0:99)

t6(x,y)=(1.0,0.0)
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 2) then
    t6(x,y-1)=(1.0,0.0)
  endif
  if(yb .eq. 3) then
    t6(x,y-1)=(-1.30902,-0.951057)
  endif
  if(yb .eq. 4) then
    t6(x,y-1)=(-1.0,0.0)
  endif
endif
else
  xb=m5(x-1)
  if(xb .eq. 2) then
    t6(x-1,y)=(1.0,0.0)
  endif
  if(xb .eq. 3) then
    t6(x-1,y)=(-1.30902,0.951057)
  endif
  if(xb .eq. 4) then
    t6(x-1,y)=(-1.0,0.0)
  endif
endif
endif
end

```

```

complex variant template t7(k)
common m2, m5
integer m2(0:99), m5(0:99)

t7(x,y)=(1.0,0.0)
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 3) then
    t7(x,y-1)=(1.0,0.0)
  endif
  if(yb .eq. 4) then
    t7(x,y-1)=(-0.809017,0.587885)
  endif
endif
else
  xb=m5(x-1)
  if(xb .eq. 3) then
    t7(x-1,y)=(1.0,0.0)
  endif
  if(xb .eq. 4) then
    t7(x-1,y)=(-0.809017,0.587885)
  endif
endif
endif
end

```

```

complex variant template t8(k)
common m2, m5
integer m2(0:99), m5(0:99)

t8(x,y)=(1.0,0.0)
if (k .eq. 1) then
  yb=m5(y-1)
  if(yb .eq. 4) then
    t8(x,y-1)=(1.0,0.0)
  endif
endif
else
  xb=m5(x-1)
  if(xb .eq. 4) then
    t8(x-1,y)=(1.0,0.0)
  endif
endif
endif
end

```



```
subroutine oemperm(A,oemp,k)
```

```
complex A(100,100)
```

```
complex z
```

```
integer oemp(100)
```

C The permutations corresponding to the partial pivots are
C executed here

```

if (k .eq. 1) then
  do 1 j=1,100
    do 5 i=100,1,-1
      if (oemp(i) .ne. i) then
        z=A(j,oemp(i))
        A(j,oemp(i))=A(j,i)
        A(j,i)=z
      endif
5      continue
1      continue
    else
      do 2 j=1,100
        do 6 i=100,1,-1
          if (oemp(i) .ne. i) then
            z=A(oemp(i),j)
            A(oemp(i),j)=A(i,j)
            A(i,j)=z
          endif
6          continue
2          continue
        endif
      endif
    end
end
```

For completeness, we include the programs used to generate the permutation tables and the multiplication images.

C Program lambda5.f

```
complex L(100),f(5),w5
complex f5
complex c0,c1,c2,c3
```

C This is a program which generates a row of the
C image corresponding to the diagonal matrix
C $I_{20} \otimes \Lambda_5$.

```
pi=3.1415926
twopi=2.0*pi
xn=5.0
arg=twopi/xn
w5=cmplx(cos(arg),sin(arg))
```

```
c0=w5-cmplx(1)
c1=(w5**3)-cmplx(1)
c2=(w5**4)-cmplx(1)
c3=(w5**2)-cmplx(1)
```

C We multiply by 0.25 because it needs to be done in order
C to compute the forward and inverse 4-pt DFT. Since lambda5
C is used in between it is a convenient place to absorb the
C multiplication.

```
f(1)=cmplx(1)
f(2)=cmplx(0.25)*f5(cmplx(1),c0,c1,c2,c3)
f(3)=cmplx(0.25)*f5(cmplx(0,-1),c0,c1,c2,c3)
f(4)=cmplx(0.25)*f5(cmplx(-1),c0,c1,c2,c3)
f(5)=cmplx(0.25)*f5(cmplx(0,1),c0,c1,c2,c3)
```

```
do 1 j=1,100
  jbar=mod(j-1,5)+1
  L(j)=f(jbar)
1 continue
```

```
open(unit=15,file='L5',status='new')
do 10 j=1,100
10 write(15,*) L(j)
close(15)
```

```
end
```

```
complex function f5(x,c0,c1,c2,c3)
```

```
complex x
complex c0,c1,c2,c3
f5=c0+x*(c1+x*(c2+x*c3))
end
```

```
C*****
```

```
C   Program d55.f
```

```
complex tw(100),w5
integer r,q
```

```
C   This is the program which generates a row of the image
C   corresponding to the diagonal matrix  $I_{20} \otimes D(5,5)$ .
```

```
pi=3.1415926
twopi=2.0*pi
xn=25.0
arg=twopi/xn
w5=cmplx(cos(arg),-sin(arg))
```

```
do 1 j=1,100
jbar = mod(j-1,25)
r=mod(jbar,5)
q=(jbar-r)/5
tw(j)=w5**(r*q)
```

```
1   continue
```

```
open(unit=15,file='twiddle',status='new')
do 10 j=1,100
10  write(15,*) tw(j)
close(15)
```

```
end
```

C Program genp1.f

```
integer s(100),so(100),r,q,qbar
```

C This is the program to generate the permutation table P1

```
do 1 j=1,100
  jbar=mod(j-1,4)
  if (jbar .eq. 1) then
    s(j)=j+1
  else
    if (jbar .eq. 2 ) then
      s(j)=j-1
    else
      s(j)=j
    endif
  endif
endif
```

1 continue

C elementary circulant

```
do 3 j=1,100
  jbar=j-1
  r=mod(jbar,4)
  q=(jbar-r)/4
  qbar=mod(q,4)
  if ( r + qbar .lt. 4 ) then
    so(j)=s(j+qbar)
  else
    q=4-qbar
    so(j)=s(j-q)
  endif
endif
```

3 continue

C shuffle 25q+r -> q+4r

```
do 2 j=1,100
  jbar=j-1
  r=mod(jbar,25)
  q=(jbar-r)/25
  k=q+4*r+1
  s(j)=so(k)
```

2 continue

```
open(unit=15,file='P1',status='new')
write(15,*)(s(j),j=1,100)
close(15)
```

end

C Program genp2.f

```
integer s(100),so(100),r,q
```

C This is the program to generate the permutation table P2

C I25 tensor P(5,5) 5q+r -> q+5r

```
do 1 j=1,25
  jbar=j-1
  r=mod(jbar,5)
  q=(jbar-r)/5
  k=q+5*r+1
  s(j)=k
  s(j+25)=k+25
  s(j+50)=k+50
  s(j+75)=k+75
```

1 continue

C shuffle 4q+r -> q+25r

```
do 2 j=1,100
  jbar=j-1
  r=mod(jbar,4)
  q=(jbar-r)/4
  k=q+25*r+1
  so(j)=s(k)
```

2 continue

```
do 3 j=1,100
  jbar=mod(j-1,4)
  if (jbar .eq. 1) then
    s(j)=so(j+1)
  else
    if (jbar .eq. 2 ) then
      s(j)=so(j-1)
    else
      s(j)=so(j)
    endif
  endif
```

3 continue

```
open(unit=15,file='P2',status='new')
write(15,*)(s(j),j=1,100)
close(15)
```

```
end
```

C Program genp3.f

```
integer s(100),so(100),r
```

C This is the program used to generate the permutation table P3

```
do 2 j=1,100
  r=mod(j-1,5)
  if ( r .eq. 2) then
    s(j)=j+1
  else
    if ( r .eq. 3 ) then
      s(j)=j-1
    else
      s(j)=j
    endif
  endif
2 continue

do 1 j=1,100
  r=mod(j-1,5)
  if ( r .eq. 0 ) so(j)=s(j)
  if ( r .eq. 1 ) so(j)=s(j+3)
  if ( r .eq. 2 ) so(j)=s(j-1)
  if ( r .eq. 3 ) so(j)=s(j)
  if ( r .eq. 4 ) so(j)=s(j-2)
1 continue

print *,(so(j),j=1,100)
open(unit=15,file='P3',status='new')
write(15,*)(so(j),j=1,100)
close(15)
end
```

C*****

C Program genp4.f

```
integer s(100),so(100),r
```

C This is the program to generate the permutation table P4

```
do 1 j=1,100
  r=mod(j-1,5)
  if ( r .eq. 0 ) s(j)=j
  if ( r .eq. 1 ) s(j)=j+2
  if ( r .eq. 2 ) s(j)=j+2
  if ( r .eq. 3 ) s(j)=j-1
  if ( r .eq. 4 ) s(j)=j-3
1 continue
```

```

do 2 j=1,100
  r=mod(j-1,5)
  if (r .eq. 2) then
    so(j)=s(j+1)
  else
    if (r .eq. 3) then
      so(j)=s(j-1)
    else
      so(j)=s(j)
    endif
  endif
2 continue

open(unit=15,file='P4',status='new')
write(15,*)(so(j),j=1,100)
end

C*****
C Program genp5.f

integer s(100),r,q

C This is the program to generate the permutation table P5
C l25 tensor P(5,5) 5q+r -> q+5r

do 1 j=1,25
  jbar=j-1
  r=mod(jbar,5)
  q=(jbar-r)/5
  k=q+5*r+1
  s(j)=k
  s(j+25)=k+25
  s(j+50)=k+50
  s(j+75)=k+75
1 continue

open(unit=15,file='P5',status='new')
write(15,*)(s(j),j=1,100)
end

C*****
C Program genp6.f

integer s(100),so(100),r,q

C This is the program to generate the permutation table P6
C Shuffle perm 25q+r -> q+4r

do 1 j=1,100

```

```

r=mod(j-1,25)
q=(j-1-r)/25
k=q+4*r+1
s(j)=k
1 continue

do 2 j=1,100
if (j .le. 25 ) then
so(j)=s(j)
else
jbar=j-1
r=mod(jbar,25)
q=((jbar-r)/25)*6
if ( r+q .lt. 25 ) then
so(j)=s(j+q)
else
q=25-q
so(j)=s(j-q)
endif
endif
2 continue

open(unit=15,file='P6',status='new')
write(15,*)(so(j),j=1,100)
close(15)

end

```


This concludes the presentation of programs used to implement some of the decompositions developed in this dissertation. The programs illustrate how oblique elimination can be used in conjunction with FFT-based methods to develop good local algorithms for computing DFT's. They also illustrate how easy it is to write a program for computing DFT's of any blocksize (up to some upper bound) locally. The latter programs also illustrate how the image algebra can be used as a basis for an algebraically based language which supports highly parallel operations.

REFERENCES

1. H. C. Andrews, *Computer Techniques in Image Processing*, Academic Press, New York, NY (1970).
2. L.A. Ankeney and G.X. Ritter, "Cellular Topology Applications in Image Processing," *Internatl. Journ. of Comp. and Inf. Science* **12**(6) (1983), 433-456.
3. L. Auslander and R. Tolmieri, "Is Computing with the Finite Fourier Transform Pure or Applied Mathematics?," *Bulletin of the AMS* **2**(6) (November, 1979), 847-894.
4. M. Behzad, G. Chartrand, and L. Lesniak-Foster, *Graphs and Digraphs*, Wadsworth International Group, Belmont, CA (1979).
5. G. Birkhoff and J. Lipson, "Heterogeneous Algebras," *J. Combinatorial Theory* **8** (1970), 115-133.
6. R. Blahut, *Fast Algorithms for Digital Signal Processing*, Addison-Wesley, Reading, MA (1985).
7. S. Chen and G. X. Ritter, "Image Processing Architectures and Languages," pp. 723-727 in *Proc. of the 1983 IEEE International Conference on Computer Design*, New York, NY (1983).
8. E. Cloud and W. Holsztynski, "Higher Efficiency for Parallel Processors," *Proceedings IEEE Southcon 84*, (March, 1984), 416-422.
9. J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Computation* **19** (April 1965), 297-301.
10. P. J. Davis, *Circulant Matrices*, John Wiley and Sons, New York, NY (1979).
11. P. Diaconis and R. L. Graham, "The discrete Radon transform on \mathbf{Z}_2^k ," *Pacific Journal of Mathematics* **118**(2) (June 1985), 323-345.
12. M. Duff, "CLIP4 A Large Scale Integrated Circuit Array Parallel Processor," in *3rd Internatl. Joint Conf. on Pattern Recognition* (1976).
13. M. Duff and S. Levialdi, edit., *Languages and Architectures for Image Processing*, Academic Press, New York, NY (1981).

14. O. Ersoy, "Semisystolic Array Implementation of Circular, Skew Circular, and Linear Convolutions," *IEEE Trans. on Computers* **C-34**(2) (February, 1985), 190-196.
15. G. H. Golub and C. F. van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, MD (1983).
16. I. J. Good, "The Interaction Algorithm and Practical Fourier Analysis," *J. Roy. Statist. Soc.* **20** (1958), 270-275.
17. I. J. Good, "The Relationship Between Two Fast Fourier Transforms," *IEEE Trans. Comp.* **C-20**(3) (March, 1971), 310-317.
18. R. W. Hamming, *Digital Filters*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1977).
19. V. E. Hill, *Groups, Representations, and Characters*, Macmillian Publishing Co., New York, NY (1975).
20. R. Hockney and C. Jesshope, *Parallel Computers : Architecture, Programming, and Algorithms*, Adam Hilger, Bristol (1981).
21. C. R. Jesshope, "The Implementation of Fast Radix Two Transforms on Array Processors," *IEEE Transactions on Computers* **C-29**(1) (January, 1980), 20-27.
22. D. Kahaner, "Matrix Description of the Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.* **AU-18** (1970), 442-450.
23. T. Kailath, B. Levy, L. Ljung, and M. Morf, "The Factorization and Representation of Operators in the Algebra Generated by Toeplitz Operators," *SIAM J. Appl. Math.* **37**(3) (December, 1979), 467-484.
24. E. Kaltofen, "Polynomial Time Reduction from Multivariate to Bivariate to Univariate Integer Polynomial Factorization," *SIAM J. Comput* **14** (1985), 469-489.
25. E. W. Kent and S. L. Tanimoto, "Hierarchical Cellular Logic and the PIPE Processor, Structural and Functional Correspondences," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, (1985), 311-320.
26. R. Keown, *An Introduction to Group Representation Theory*, Academic Press, New York, NY (1975).
27. D. E. Knuth, *The Art of Computer Programming, Sorting and Searching*, Addison-Wesley, Reading, MA (1973).

28. W. Ledermann, *Introduction to Group Characters*, Cambridge University Press, Cambridge (1977).
29. R. M. Loughheed and D. L. McCubbrey, "The Cytocomputer : A Practical Pipelined Image Processor," *Proceedings of the Seventh International Symposium on Computer Architecture*, (1980), 411-418.
30. J. H. McClellan and C. M. Rader, edit., *Number Theory in Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J. (1979).
31. P.E. Miller, "An Investigation of Boolean Image Neighborhood Transformations," Ph.D thesis, Ohio State University (1978).
32. P. E. Miller, "Development of a Mathematical Structure for Image Processing," *Perkin-Elmer Optical Division Tech. Report*, (1983).
33. D. R. Musser, "Multivariate Polynomial Factorization," *Journal of the ACM* **22**(2) (April, 1975), 291-308.
34. J. von Neumann, *Theory of Self-Reproducing Automata*, University of Illinois Press, Urbana, IL (1966).
35. D. E. Oldfield and S. F. Reddaway, "An Image Understanding Performance Study on the ICL Distributed Array Processor," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, (1985), 256-265.
36. A. V. Oppenheim and R. W. Schaffer, *Digital Signal Processing*, Prentice-Hall Inc., Englewood Cliffs, NJ (1975).
37. J. M. Ortega and W. G. Poole, *An Introduction to Numerical Methods for Differential Equations*, Pittman Publishing Inc., Marshfield, MA (1981).
38. B. N. Parlett, "Winograd's Fourier Transform via Circulants," *Linear Algebra and Its Applications* **45** (1982), 137-155.
39. M. C. Pease, "An Adaptation of the Fast Fourier Transform for Parallel Processing," *Journal ACM* **15**(2) (April, 1968), 253-264.
40. J. L. Potter, "Image Processing on the Massively Parallel Processor," *Computer* **16**(1) (January, 1983), 62-68.
41. W. K. Pratt, "A Pipeline Architecture for Image Processing and Analysis," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, (1985), 516-521.

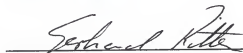
42. W. Rheinboldt, Chairman, *Future Directions in Computational Mathematics, Algorithms, and Software*, Society for Industrial and Applied Mathematics, Philadelphia, PA (1985).
43. G. X. Ritter, "An Algebra of Images," *Proc. Conf. on Intelligent Systems and Machines*, (1984), 333-339.
44. G. X. Ritter, "The Language of Massively Parallel Processing Computers," *Proc. 1984 S. E. Reg. ACM Conference, Friendly Systems:1984-2001?*, (1984), 224-231.
45. G.X. Ritter and P.D. Gader, "Image Algebra Implementation on Cellular Array Computers," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, (1985), 430-438.
46. G. X. Ritter, P. D. Gader, and J. L. Davidson, "Automated Bridge Detection in FLIR Images," in *Proceedings of the Eighth International Conference on Pattern Recognition*, Paris, France (to appear).
47. D. J. Rose, "Matrix Identities of the Fast Fourier Transform," *Linear Algebra and Its Applications* **29** (1980), 423-443.
48. A. Rosenfeld, "Parallel Image Processing Using Cellular Array Computers," *Computer* **1**(1) (Jan. 1983), 177-191.
49. U. A. Rouhala, "Introduction to Array Algebra," *J. of Photogrammetric Engineering and Remote Sensing* **46**(2) (February, 1980), 177-191.
50. J. T. Schwartz, "Mathematics Addresses Problems in Computer Vision for Advanced Robots," *SIAM News* **18**(3) (1985), 3.
51. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London (1982).
52. H. B. Sexton, "Cayley Networks as Parallel Computer Architectures," pp. 12 in *Cooperative Research Associateships tenable at the Naval Ocean Systems Center*, National Research Council, Washington, DC (1986).
53. T. M. Silverberg, "The Hough Transform on the Geometric Arithmetic Array Processor," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, (Nov. 1985), 387-394.
54. S. Sternberg, "Cellular Computers and Biomedical Image Processing," in *Proc. U.S. - France Seminar on Biomed. Image Processing*, Grenoble, France (1980).

55. S. R. Sternberg, "Overview of Image Algebra and Related Issues," in *Integrated Technology for Parallel Image Processing*, edit. S. Levialdi, Academic Press, London (1985).
56. S. R. Sternberg, "Automatic Image Processor," *U. S. Patent 4,167,728*, (Sept. 11, 1979).
57. S. R. Sternberg, "Languages and Architectures for Parallel Image Processing," in *Pattern Recognition in Practice*, North-Holland Publishing Co., New York, NY (1980).
58. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer Verlag, New York, NY (1980).
59. J. P. Strong, "The Fourier Transform on Mesh Connected Arrays Such as the Massively Parallel Processor," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, (Nov. 1985), 190-197.
60. M. Tchuente, "Parallel Calculation of a Linear Mapping on a Computer Network," *Linear Algebra and Its Applications* **28** (1979), 223-247.
61. M. Tchuente, "Parallel Realization of Permutations Over Trees," *Discrete Math.* **39** (1982), 211-214.
62. F. Theilheimer, "A Matrix Version of the Fast Fourier Transform," *IEEE Trans. Audio Electroacoust.* **AU-17** (1969), 158-161.
63. L. Uhr, *Algorithm-Structured Computer Arrays and Networks*, Academic Press, New York (1984).
64. S. M. Ulam, "On Some New Possibilities in the Organization and Use of Computing Machines," *IBM Research Report RC68* (May, 1957).
65. J. S. Wiegak, H. Buxton, and B.F. Buxton, "Convolution with Separable Masks for Early Image Processing," *Computer Vision, Graphics, and Image Processing* **32** (1985), 279-290.
66. S. Winograd, "On Computing the Discrete Fourier Transform," *Math. Comp.* **32** (January, 1978), 175-199.

BIOGRAPHICAL SKETCH

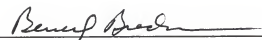
Paul Gader was born September 11, 1956, in Jamestown, North Dakota. He lived in North Dakota until he joined the U. S. Navy at the age of seventeen. He was honorably discharged after three years, at which time he moved to Orlando, Florida. While there, he attended Valencia Community College and received his B. S. degree in mathematics at the University of Central Florida. He received his M. S. degree in mathematics at the University of Florida in 1983. He was married to Elizabeth Dunn in 1984.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



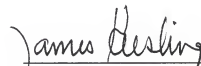
Gerhard X. Ritter, Chair
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Beverly Brechner
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



James Keesling
Professor of Mathematics

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Maurice Shrader-Frechette
Assistant Professor of Computer and Information Sciences

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David C. Wilson

David Wilson
Associate Professor of Mathematics

This dissertation was submitted to the Graduate Faculty of the Department of Mathematics in the College of Liberal Arts and Sciences and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August 1986

Dean, Graduate School